ECHELON®

OpenLNS
Programmer's Reference

078-0437-01A

Printed in the United States of America.

Echelon Corporation
www.echelon.com

Table of Contents

Preface

This document provides an overview of how to use the OpenLNS Object Server Active X control, and it contains detailed information on each object, property, method, event and constant you will use when writing OpenLNS applications. This includes information such as the syntax you should use when calling a method or writing to a property, and the version when each item was Added to API.

Purpose

This document provides reference information for writing OpenLNS applications.

Audience

This guide is intended for software developers creating OpenLNS applications. OpenLNS applications may be written in any language that supports COM Components or ActiveX controls, including Microsoft® Visual C# .NET, Visual C++ .NET, and Visual Basic .NET. Readers of this guide should have experience programming in these languages, and be familiar with LONWORKS® technology and COM/ActiveX control concepts.

System Requirements

The following section lists the system requirements for computers running the OpenLNS Standard Developer's Kit (SDK) and the OpenLNS Server (requirements for smaller and larger systems are listed separately).

OpenLNS SDK

System requirements for computers running the OpenLNS SDK are as follows:

- Windows 7 (64-bit and 32-bit), Windows Server 2008 SR2 64-bit and 32-bit (64-bit RNI required; U10/U20 desirable), Windows Vista with Service Pack (SP) 1, or Windows XP with SP3 (32-bit).
- 500 MHz processor or faster.
- 2 GB or more of free disk space.
- 512 MB RAM.
- 1,024 MB page file minimum.
- Microsoft Visual Studio 2010 or Microsoft Visual Studio 2008 (.NET Framework 3.5) for development of Visual C# .NET, Visual C++ .NET, and Visual Basic .NET applications.
- DVD-ROM drive
- 1,024 x 768 or higher-resolution display with at least 256 colors.
- Mouse or compatible pointing device
- Local, remote, or IP-852 OpenLDV 4.0 network interface.
    - Compatible local network interfaces include the U10/U20 USB network interface; PCC-10, PCLTA-20, or PCLTA-21 network interface cards; and the SLTA-10 Serial LonTalk Adapter. The PCC/PCLTA and SLTA-10 network interfaces are compatible with 32-bit versions of Windows only.
    - Compatible remote network interfaces include the SmartServer, i.LON 100 *e*3 Internet Server, i.LON 600 LONWORKS/IP-852 Router, and i.LON 10 Ethernet Adapter.
    - Compatible IP-852 network interfaces include the SmartServer, i.LON 100 *e*3 Internet Server, and i.LON 600 LONWORKS/IP-852 Router.

OpenLNS Server Computer (Smaller Network)

System requirements for computers running the OpenLNS Server on a smaller network are as follows:

- Windows 7 (64-bit and 32-bit), Windows Server 2008 SR2 64-bit and 32-bit (64-bit RNI required; U10/U20 desirable), Windows Vista with Service Pack (SP) 1, or Windows XP with SP3 (32-bit).
- 100MHz processor or faster
- 50 MB or more of free disk space. This does not account for the size of the OpenLNS application or the OpenLNS network databases.
- 256 MB RAM or more. This may vary depending on the requirements of the OpenLNS Server and the OpenLNS applications running on the computer.
- 1,024 MB page file minimum.
- Microsoft Visual Studio 2010 or Microsoft Visual Studio 2008 (.NET Framework 3.5) for development of Visual C# .NET, Visual C++ .NET, and Visual Basic .NET applications.
- DVD-ROM drive
- 1,024 x 768 or higher-resolution display with at least 256 colors.
- Mouse or compatible pointing device
- Local, remote, or IP-852 OpenLDV 4.0 network interface.
    - Compatible local network interfaces include the U10/U20 USB network interface; PCC-10, PCLTA-20, or PCLTA-21 network interface cards; and the SLTA-10 Serial LonTalk Adapter. The PCC/PCLTA and SLTA-10 network interfaces are compatible with 32-bit versions of Windows only.
    - Compatible remote network interfaces include the SmartServer, i.LON 100 *e*3 Internet Server, i.LON 600 LONWORKS/IP-852 Router, and i.LON 10 Ethernet Adapter.
    - Compatible IP-852 network interfaces include the SmartServer, i.LON 100 *e*3 Internet Server, and i.LON 600 LONWORKS/IP-852 Router.

OpenLNS Server Computer (Larger, Busier Network)

System requirements for computers running the OpenLNS Server on a larger network are as follows (these requirements are also valid for computers on which a large network database is engineered prior to the network's OpenLNS Server becoming operational):

- Windows 7 (64-bit and 32-bit), Windows Server 2008 SR2 64-bit and 32-bit (64-bit RNI required; U10/U20 desirable), Windows Vista with Service Pack (SP) 1, or Windows XP with SP3 (32-bit).
- 2GHz processor or faster
- 50 MB or more of free disk space. This does not account for the size of the OpenLNS application or the OpenLNS network databases. Use a high-performance hard disk for the best performace during development.
- 2 GB RAM or more. This may vary depending on the requirements of the OpenLNS Server and the OpenLNS applications running on the computer.
- 2,048 MB page file minimum.

- Microsoft Visual Studio 2010 or Microsoft Visual Studio 2008 (.NET Framework 3.5) for development of Visual C# .NET, Visual C++ .NET, and Visual Basic .NET applications.
- DVD-ROM drive
- 1,024 x 768 or higher-resolution display with at least 256 colors.
- Mouse or compatible pointing device
- Local, remote, or IP-852 OpenLDV 4.0 network interface.
    - Compatible local network interfaces include the U10/U20 USB network interface; PCC-10, PCLTA-20, or PCLTA-21 network interface cards; and the SLTA-10 Serial LonTalk Adapter. The PCC/PCLTA and SLTA-10 network interfaces are compatible with 32-bit versions of Windows only.
    - Compatible remote network interfaces include the SmartServer, i.LON 100 *e*3 Internet Server, i.LON 600 LONWORKS/IP-852 Router, and i.LON 10 Ethernet Adapter.
    - Compatible IP-852 network interfaces include the SmartServer, i.LON 100 *e*3 Internet Server, and i.LON 600 LONWORKS/IP-852 Router.

OpenLNS Documentation

The documentation for OpenLNS is provided as Adobe Acrobat PDF files and online help files. The PDF file for this document is installed in the **Echelon OpenLNS Utilites** program folder when you install the Echelon OpenLNS software. You can also download the latest OpenLNS documentation, including the latest version of this guide, by going to the Echelon OpenLNS Web site at *www.echelon.com/openlns*.

OpenLNS CT User's Guide	Describes how to use the OpenLNS Commissioning Tool (CT) to design, commission, modify, and maintain LONWORKS networks.
OpenLNS Programmer's Guide	Describes how to use the OpenLNS Object Server ActiveX Control to develop OpenLNS apps.
OpenLNS® Plug-in Framework Developer's Guide	Describes how to write OpenLNS system and device plug-ins using .NET programming languages such as C# and Visual Basic .NET.

The following documents supplement the material provided in this guide. You can download these documents from Echelon's Web site at *www.echelon.com/docs*.

Introduction to the LONWORKS® Platform	Provides a high-level introduction to LONWORKS networks and the tools and components that are used for developing, installing, operating, and maintaining them.

For More Information and Technical Support

The **Echelon OpenLNS Utilties ReadMe** document provides descriptions of known problems, if any, and their workarounds. To view the **Echelon OpenLNS Utilties ReadMe** document, click **Start**, point to **Programs**, point to **Echelon OpenLNS Utilities**, and then select **Echelon OpenLNS Utilities ReadMe**.

If you have technical questions that are not answered by this document, the online help files provided with the *OpenLNS Standard Developer's Kit*, or the **Echelon OpenLNS Utilties ReadMe** 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.

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*

Content

This guide includes the following content:

- *OpenLNS Object Server*. Introduces the OpenLNS Object Server, describes the OpenLNS Object model, hierarchy, and naming conventions, and it
- *Objects*. Describes the objects in the OpenLNS Object hierarchy, and details their methods, properties, and events.
- *Interfaces*. Details each interface in the OpenLNS Object hierarchy.
- *OpenLNS Errors*. Lists and describes the OpenLNS errors that may be generated by the various OpenLNS components.
- *Appendix A: Deprecated Items*. Lists methods, properties, and objects that should no longer be used in OpenLNS.

1

OpenLNS Object Server

This chapter introduces the OpenLNS Object Server, describes the OpenLNS Object model, hierarchy, and naming conventions.

Introduction

The OpenLNS Object Server ActiveX Control provides high-level services for installing, diagnosing, maintaining, and monitoring and controlling LONWORKS networks. This ActiveX control is an OpenLNS client application interface into the OpenLNS network operating system. The OpenLNS network operating system allows LONWORKS network tools to interoperate with one another, and interact with the managed LONWORKS network. OpenLNS brings the power of client-server architecture, and object-oriented, component-based software design into control networks. This enables OpenLNS tools to work together to install, maintain, monitor, and control LONWORKS networks. In addition, it provides the fastest, most efficient way to bring control on-line with all your other information systems.

The OpenLNS Object Server provides its services through a set of objects which correspond to components within a LONWORKS network. These objects are therefore referred to as LONWORKS Component Architecture (LCA) objects. The classes which define the LCA objects are organized into the *OpenLNS Object Hierarchy*. The LcaObjectServer class is the highest level (root) class within this hierarchy. The root class contains lower level objects, and the lower level objects contain additional objects, creating a tree structure.

An object is "contained" by a higher level, or parent, object when it may be accessed through a property of the parent. Some properties return individual objects, while others contain a collection of objects. Lower level objects may appear in several locations within the class tree. For example, Interface objects are contained by both DeviceTemplate and *AppDevice* objects. See the *OpenLNS Object Server Object Model* for more information.

OpenLNS Object Server Object Model

This section gives an overview of the object model depicted in the *OpenLNS Object Hierarchy*. The "Lca" prefixes have been eliminated from the names for this discussion for reasons discussed in *Object Naming Convention*.

The *ObjectServer* class is the highest level (root) class within this hierarchy. An instance of this class can be created directly by an OpenLNS application. Typically, in development tools such as Visual C# .NET, Visual C++ .NET, and Visual Basic .NET, this is done by dragging a representation of the control from a palette onto a form or workspace. This creates an ObjectServer object, which represents an instance of the OpenLNS ActiveX control.

Many objects include properties that reference other objects or collections of objects. For example, an *AppDevice* object contains a *DeviceTemplate* object which references the device template which was used to create the application device that the *AppDevice* object represents. These object references define a tree structure for the Object Server.

The ObjectServer object directly references six *collection objects*: the *ComponentApps* class (in the ObjectServer object's *ComponentApps* property), the *NetworkInterfaces* class (in the *NetworkInterfaces* property), three instances of the *Networks* class (in the *Networks*, *RemoteNetworks*, and *VNINetworks* properties), and the *Extensions* class (in the *Extensions* property). These objects represent the component applications, network interfaces, extensions, and networks which have been registered on the computer or are otherwise available for use by the Object Server.

The remainder of the OpenLNS Objects originate from the three Networks collections. A network is represented by an instance of the Network object, which contains both a systems and channels collection. A network defines a physical LONWORKS network, while a system is an entity, both logical and physical, that uses the network fabric. Each Network object has a *MyVNI* property which contains an *AppDevice* object which contains the *MonitorSets* collection for that network. Each *MonitorSet* contains an *NvMonitorPoints* collection and an *MsgMonitorPoints* which represent the network variable and message monitor points contained in that monitor set.

One *System* object is supported per network. *Channels* are physical objects which may be shared by multiple systems (each with their own independent OpenLNS Server); they are accessible via the Network object.

The *System* object has a tree of objects below it. A *TemplateLibrary* object has the role of a profile or "parts catalog", containing generic parts that can be applied across different objects. For example, a single *HardwareTemplate* object or *DeviceTemplate* object can be defined that is later associated with more than one *AppDevice*. A *HardwareTemplate* contains information such as the hardware type and Neuron model of a device and is used only for devices with application images which are built from source code using the LCA Field Compiler. A *DeviceTemplate* contains all the information necessary to define a generic *AppDevice*; the information contained in the DeviceTemplate can come from the external interface files (.XIF, .XFB, and .XFO extensions), a source program, or the device itself. A *ConnectDescTemplate* object describes a generic connection, including its LonTalk protocol service and other connection attributes.

The *System* object contains a *Subsystems* collection. *Subsystems* are used to organize devices similar to how directories are used to organize files—they could represent groupings of devices in a room, for instance. The concept of a location in the OpenLNS Commissioning Tool (CT) can be represented using a *Subsystem*. By allowing nested *Subsystems*, the *Object Server* makes it possible to define subsystem hierarchies. For example, a building subsystem may consist of floor subsystems, each of which consists of room subsystems, each of which consist of HVAC, security, and lighting subsystems.

A single device may be associated with multiple subsystems, and must be associated with at least one. For example, a VAV controller may appear in both a Floor subsystem and an HVAC subsystem. When initially defining a device, it is first added to a single subsystem. References to the device may then be added to other subsystems. The device is not deleted from the OpenLNS database or decommissioned until all references have been deleted. The device's association with the first subsystem is also treated as a reference, so it may be removed from its initial subsystem at any time.

The *AppDevices* collection, and the *AppDevice* objects that are part of it, are key components of a System. *AppDevice* objects represent individual application devices (also called nodes). Both Neuron Chip-hosted and host-based devices are represented by this object.

Each *AppDevice* object contains an *Interface* object. The *Interface* object contains *LonMarkObject*, *NetworkVariable*, *ConfigProperty*, and *MessageTag* objects that define the external interface to the device. *AppDevices* may also contain an *Interfaces* collection object, if the underlying physical device supports dynamic network variables.

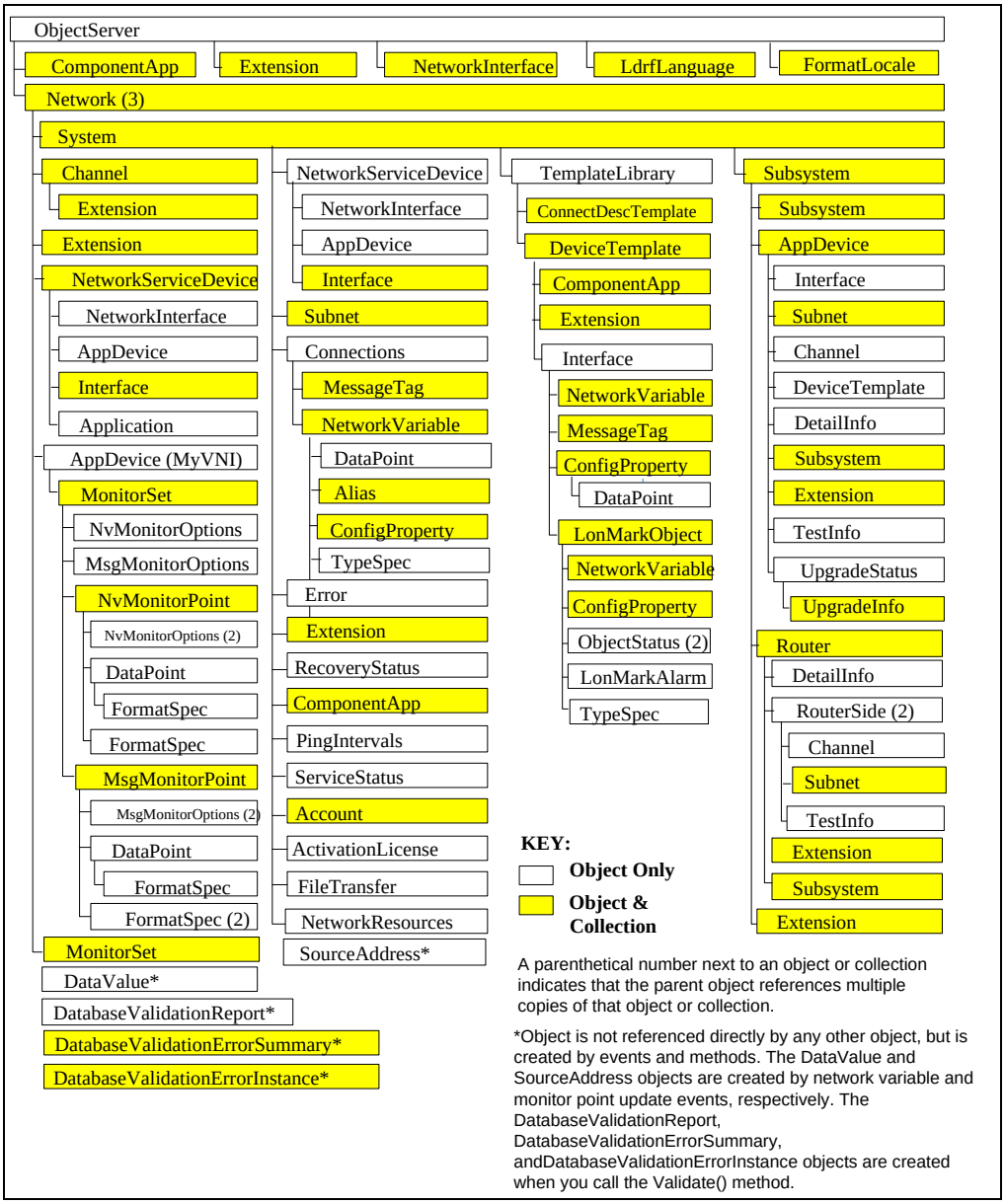
DeviceTemplate objects also contain an Interface object that defines the external interface for the device template. The *DeviceTemplate* object's Interface has the same definition as the *AppDevice* object's Interface.

User-defined object names must be unique within their respective collections. For example, *Subsystem* names, *AppDevice* names, and *Router* names must be unique within

a *Subsystem*. Different *Subsystems* can contain devices with the same name, however. For example, a building may consist of a Subsystem object for each room, with each room Subsystem including *AppDevice* objects called VAV and Thermostat.

The *NetworkServiceDevice* class provides a logical representation of the OpenLNS Server and its NSI. The network service device, in turn, contains a *NetworkInterface* object. This network interface determines the driver name to be associated with the system, if any. Many additional object classes are provided to OpenLNS applications through the Object Server.

OpenLNS Object Hierarchy



Object Naming Convention

While the documentation currently references most objects by their base names, it is important to note that this name is not the full class name. The full name, which must be used when declaring object types, includes the "Lca" prefix.

For example,

Base Name	**Full Class Name**
AppDevice	LcaAppDevice
Network	LcaNetwork
Subsystems	LcaSubsystems

The use of the prefix is necessary to avoid conflicts with other ActiveX controls and servers, since the ActiveX software architecture uses a single name space for all controls and servers.

2

Objects

This chapter describes the objects in the OpenLNS Object hierarchy, and details their properties, methods, and events.

Account

Description	Represents an OpenLNS licensing account. Reserved for future use.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Accounts* collection object *System* object
Default Property	*Name* property.
Methods	None
Properties	• *AccountNumber* • *Charges* • *ClassID* • *Description* • *Name*

Methods

The *Account* object does not contain any methods.

Properties

The *Account* object contains the following properties:

- *AccountNumber*
- *Charges*
- *ClassID*
- *Description*
- *Name*

AccountNumber

Summary	Contains the account number for the specified account. Reserved for future use. This property and the associated *Account* and *Accounts* objects are reserved for future use
Availability	Local, full, and lightweight clients.
Syntax	*acctNumber* = *acctObject*.**AccountNumber** **Element** — **Description** *acctNumber* — A Long containing the account number. *acctObject* — The *Account* object to be acted on.
Data Type	Long.
Read/Write	Read and write.
Added to API	Prior to LNS Release 3.0.

Charges

Summary	Contains the number of charges applied to the specified account. Reserved for future use. This property allows the object type to be determined when it is unknown (for

	example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, and lightweight clients.
Syntax	*numCharges* = *acctObject*.**Charges** **Element** **Description** *numCharges* A Long containing the number of charges. *acctObject* The *Account* object to be acted on.
Data Type	Long.
Read/Write	Read and write.
Added to API	Prior to LNS Release 3.0.
Comments	This property and the associated *Account* and *Accounts* objects are reserved for future use.

ClassID

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Account* object in the *ConstClassIds* constant: **54 lcaClassIdAlias** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Description

Summary	Stores description information about the object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description**

	stringValue	A string description of the object.
	object	The object to be acted on.
Data Type	Long.	
Read/Write	Read and write.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	
Comments	This property is read only for *Error* objects.	

Name

Summary	Specifies the name of an object as a character string.	
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.	
Syntax	*stringValue* = *object*.**Name**	
	Element	**Description**
	stringValue	The name of the object.
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read/write.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Accounts

The following table summarizes the *Accounts* object.

Description	Represents an OpenLNS licensing account. Reserved for future use.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object
Default Property	*Item* property.
Methods	• *Add* • *Remove*
Properties	• *ClassID* • *Count* • *Item* • *_NewEnum*

Methods

The *Account* object contains the following methods:

- *Add*
- *Remove*

Add

Summary	Adds an object to the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Add** *name* **Element** — **Description** *objectColl* — The collection being acted upon. *name* — A String value specifying the name of the object being added.
Added to API	Prior to LNS Release 3.0.

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed. *name* — A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *Account* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Accounts* object in the *ConstClassIds* constant: **55 lcaClassIdAccounts** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property is read only for *Error* objects.

Item

Summary	Returns an object from a collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*)

	Element	**Description**
	retrievedObject	The object retrieved from the collection.
	collObject	The collection object to be acted on.
	index	A Long type specifying the ordinal index of the object to retrieve.
	stringExpression	A string type specifying the name of the object to retrieve.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

ActivationLicense

An *ActivationLicense* object represents a snapshot of the current activation license and related information on the OpenLNS Server at the time it was first accessed. The following table summarizes the *ActivationLicense* object.

Description	A snapshot of the current activation license on the OpenLNS Server.
Added to API	OpenLNS.
Accessed Through	*System* object.
Default Property	None.
Methods	• *Refresh*
Properties	• *ActivatedVersion* • *ClassId* • *DaysRemaining* • *DeviceCapacity* • *DeviceCapacityConsumed* • *ExpirationDate* • *LicesnseId* • *LicenseStatus* • *IsTrialLicense* • *MaxOpenSystems* • *RequiredVersion* • *RunTimeLimit* • *RunTimeRemaining*

Methods

The *ActivationLicense* object contains the following method.

Refresh

Summary	Updates the *ActivationLicense* object with a fresh snapshot of the activation license.
Availability	Local, full, and lightweight clients.
Syntax	*ActivationLicense*.**Refresh** **Element** — **Description** *ActivationLicense* — The *ActivationLicense* object to be acted on.
Added to API	OpenLNS.

Properties

The *ActivationLicense* object contains the following properties:

- *ActivatedVersion*
- *ClassId*
- *DaysRemaining*

- *DeviceCapacity*
- *DeviceCapacityConsumed*
- *ExpirationDate*
- *LicesnseId*
- *LicenseStatus*
- *IsTrialLicense*
- *MaxOpenSystems*
- *RequiredVersion*
- *RunTimeLimit*
- *RunTimeRemaining*

ActivatedVersion

Summary	The version number that the license covers. If there is no valid license, this value is an empty string.
Availability	Local, full, and lightweight clients.
Syntax	*version* = *alObject*.**ActivatedVersion** **Element** — **Description** *version* — The version number that the license covers. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	OpenLNS.

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Alias* object in the *ConstClassIds* constant: **97 lcaClassIdActivationLicense** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

DaysRemaining

Summary	Indicates the number of days remaining before the OpenLNS Server license expires.
Availability	Local, full, and lightweight clients.
Syntax	*days* = *alObject*.**DaysRemaining** **Element** — **Description** *days* — The number of days until the license expires. A value of **lcaLicenseNoRestriction** (-1) indicates that there is no expiration. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	OpenLNS.

DeviceCapacity

Summary	The total number of devices that can be defined in the database, excluding NSDs and routers. A device that has been added without specifying both a device template and a channel is not considered "defined" until that device has been commissioned.
Availability	Local, full, and lightweight clients.
Syntax	*numberDevices* = *alObject*.**DeviceCapacity** **Element** — **Description** *numberDevices* — The number of devices that can be defined in the database. A value of **65535** indicates that the database has unlimited capacity. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	OpenLNS.

DeviceCapacityConsumed

Summary	The total number of devices that have been defined in the

	database, excluding NSDs and routers. A device that has been added without specifying both a device template and a channel is not considered "defined" until that device has been commissioned.
Availability	Local, full, and lightweight clients.
Syntax	*numberDevices* = *alObject*.**DeviceCapacityConsumed** **Element** — **Description** *numberDevices* — The number of devices that have been defined in the database. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	OpenLNS.

ExpirationDate

Summary	The date on which the license expires. The license will expire at 00:00 local time on the morning of the specified date.
Availability	Local, full, and lightweight clients.
Syntax	*date* = *alObject*.**ExpirationDate** **Element** — **Description** *date* — The date the license expires. An empty string indicates that there is no expiration date for the license. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	Date.
Read/Write	Read only.
Added to API	OpenLNS.

LicenseId

Summary	The unique LNS activation license ID.
Availability	Local, full, and lightweight clients.
Syntax	*licenseIdValue* = *alObject*.**LicenseId** **Element** — **Description** *licenseIdValue* — The LNS activation license ID. This is an empty string if no license ID is found. *alObject* — The *ActivationLicense* object to be acted on.

Data Type	String.
Read/Write	Read only.
Added to API	OpenLNS.

LicenseStatus

Summary	Indicates whether the activation license is valid, and indicates why an activation license is invalid.
Availability	Local, full, and lightweight clients.
Syntax	*stateValue* = *alObject*.**LicenseStatus** **Element** **Description** *stateValue* The enumerated values for this element, which are contained in the *ConstActivationLicenseStatus* constant, are as follows: **0 lcaActivationLicenseStatusActivated** The license is valid. **1 lcaActivationLicenseStatusInvalid** The license file is invalid. **2 lcaActivationLicenseStatusWrongPc** The license file not valid for this computer. **3 lcaActivationLicenseStatusExpired** The license has expired. **4 lcaActivationLicenseStatusInvalidVersion** The license is valid, but not for this release. **5 lcaActivationLicenseStatusTimedOut** The RunTimeLimit has run out. *alObject* The *ActivationLicense* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	OpenLNS.

IsTrialLicense

Summary	Indicates whether the activation license is a trial license.
Availability	Local, full, and lightweight clients.
Syntax	*trialLicenseValue* = *alObject*. **IsTrialLicense** **Element** **Description** *trialLicenseValue* A True of False value indicating

		whether the activation license is a trial license. **TRUE**. The activation license is a trial license. **FALSE**. The activation license is not a trial license.
	alObject	The *ActivationLicense* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	OpenLNS.	

MaxOpenSystems

Summary	Indicates the maximum number of local systems that can be opened at a given time on this computer.	
Availability	Local, full, and lightweight clients.	
Syntax	*numberSystems* = *alObject*. **MaxOpenSystems**	
	Element	**Description**
	numberSystems	The maximum number of local systems that can be opened on the computer. A value of **lcaLicenseNoRestriction (-1)** indicates that there is no limit.
	alObject	The *ActivationLicense* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	OpenLNS.	

RequiredVersion

Summary	The minimum version number that is required by the current release.	
Availability	Local, full, and lightweight clients.	
Syntax	*VersionNumber* = *alObject*. **RequiredVersion**	
	Element	**Description**
	timeValue	The minimum version number required by the current release. An empty string indicates that there is no version requirement.
	alObject	The *ActivationLicense* object to be acted on.

Data Type	String.
Read/Write	Read only.
Added to API	OpenLNS.

RunTimeLimit

Summary	The maximum number of minutes that the application can use LNS after opening the first network.
Availability	Local, full, and lightweight clients.
Syntax	*timeValue* = *alObject*. **RunTimeLimit** **Element** — **Description** *timeValue* — The maximum number of minutes that the application can use LNS after opening the first network. A value of **lcaLicenseNoRestriction (-1)** indicates that there is no runtime limit. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	OpenLNS.

RunTimeRemaining

Summary	The number of minutes that the application can continue to use LNS after opening the first network.
Availability	Local, full, and lightweight clients.
Syntax	*timeLimitValue* = *alObject*. **RunTimeLimit** **Element** — **Description** *timeValue* — The number of minutes that the application can continue to use LNS after opening the first network. A value of **lcaLicenseNoRestriction (-1)** indicates that there is no runtime limit. *alObject* — The *ActivationLicense* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	OpenLNS.

Alias

The following table summarizes the *Alias* object.

Description	Represents an alias in-use for a network variable on an application device.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Aliases* collection object
Default Property	*Index* property.
Methods	None.
Properties	• *ClassID* • *Index* • *Parent* • *Selector*

Methods

The *Alias* object does not contain any methods.

Properties

The *Alias* object contains the following properties:

- *ClassID*
- *Index*
- *Parent*
- *Selector*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Alias* object in the *ConstClassIds* constant: **58 lcaClassIdAlias** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Index

Summary	Returns the index within an application device of the network variable alias.
Availability	Local, full, and lightweight clients.
Syntax	*index* = *object*.**Index** **Element** — **Description** *object* — The object to be acted on. *index* — Index of the message tag or network variable.
Valid Values	**Property** — **Values** *Message tags* — 0–14 *Neuron-hosted applications* — 0–62 *Host-based applications* — 0–8,191 *msg_in* message tag — MSG_IN_TAG (-2)
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

Selector

Summary	The network variable selector value assigned to this network variable alias. When a device is installed, selector values that represent unbound network variables are assigned to the network variables in that device. When placing the network variable in a connection, the OpenLNS Object Server assigns a value representing that connection.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*selectorValue* = *object*.**Selector** **Element** — **Description** *object* — The *Alias* object to be acted on. *selectorValue* — The network variable selector value
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Aliases

An *Aliases* object is a collection of *Alias* objects. The collection contains an *Alias* object for each alias in-use for the parent network variable. Unused alias entries contained within an application device's alias table will not appear in an *Aliases* collection.

The *Aliases* object is not updated as changes occur on the network, all property values are cached on the OpenLNS Server when the object is created. As a result, an *Aliases* object must be periodically refreshed by invoking the *Refresh* method, or by fetching a new *Aliases* collection object from the parent *NetworkVariable* object. The following table summarizes the *Aliases* object.

Description	Represents a collection of *Alias* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*NetworkVariable* object.
Default Property	*Item* property.
Methods	• *ItemByIndex* • *Refresh*
Properties	• *ClassID* • *Count* • *Item*

	• *Parent* • *_NewEnum*

Methods

The *Aliases* object contains the following methods.

- *ItemByIndex*
- *Refresh*

ItemByIndex

Summary	Retrieves an *Alias* object from an *Aliases* collection. The *Alias* object to be retrieved must be specified by its index value.
Availability	Local, full, and lightweight clients.
Syntax	*itemObject* = *itemsColl*.**ItemByIndex** *index* **Element** — **Description** *itemObject* — The *Alias* object retrieved from the collection. *itemsColl* — The *Aliases* collection to be acted on. *index* — An Integer value specifying the *Index* property of the *Alias* object to be retrieved.
Added to API	LNS Release 3.0.

Refresh

Summary	Causes the OpenLNS Server to read the alias table of the affiliated application device for aliases belonging to the ("parent") network variable. Any previously obtained *Alias* objects are not updated, and their values should be ignored and the objects released.
Availability	Local, full, and lightweight clients.
Syntax	*aliasesColl*.**Refresh** **Element** — **Description** *aliasesColl* — The *Aliases* collection object to be acted on.
Added to API	Prior to LNS Release 3.0.

Properties

The *Aliases* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Aliases* object in the *ConstClassIds* constant: **59 lcaClassIdAliases** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an object from a collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item*

	property. Index values start at 1.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The object retrieved from the collection. *collObject* — The collection object to be acted on. *index* — A Long type specifying the ordinal index of the object to retrieve. *stringExpression* — A string type specifying the name of the object to retrieve.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++,

	you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

AppDevice

An *AppDevice* object represents a LONWORKS application device, including both Neuron-hosted and host-based applications. The following table summarizes the *AppDevice* object.

Description	Represents a LONWORKS application device.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevices* collection object. *NetworkServiceDevice* object.
Default Property	*Name* property.
Methods	• *ClearStatus* • *Commission* • *CommissionEx* • *Decommission* • *Delay* • *DownloadConfigProperties* • *GetMessagePoint* • *Load* • *MoveEx* • *PostMove* • *PreMove* • *PropagateDeviceConfigUpdates* • *Reboot*

	• *ReleasePendingUpdates* • *Replace* • *ReplaceEx* • *ReplaceEx* • *Reset* • *ResyncToTemplate* • *Test* • *Upgrade* • *UploadConfigProperties* • *Wink*
Properties	• *AliasCapacity* • *AliasUseCount* • *AppImagePath* • *AttachmentStatus* • *AuthenticationEnabled* • *BitmapFilePath* • *Channel* • *ClassId* • *CommissionStatus* • *ConfigurationState* • *ConnectionUpdateType* • *Description* • *DetailInfo* • *DeviceTemplate* • *Extensions* • *Handle* • *HasBeenCommissioned* • *IconFilePath* • *InitialAuthenticationKey* • *Interface* • *Interfaces* • *LastTestInfo* • *LastUpgrade Status* • *Location* • *LocationInNeuron* • *MonitorSets* • *MtHubs* • *Name* • *NetworkServiceDevice* • *NeuronId* • *NodeId* • *NonGroupRcvTimer* • *NsiHandle* • *NVHubs* • *Parent* • *PingClass* • *PendingNeuronId* • *Priority* • *ProgramId* • *SelfDocumentation* • *State* • *SubnetId*

	• *Subnets* • *Subsystems* • *Upgrade Requirement*

Methods

The *AppDevice* object contains the following methods.

- *ClearStatus*
- *Commission*
- *CommissionEx*
- *Decommission*
- *DownloadConfigProperties*
- *GetMessagePoint*
- *Load*
- *MoveEx*
- *PostMove*
- *PreMove*
- *PropagateDeviceConfigUpdates*
- *Reboot*
- *ReleasePendingUpdates*
- *Replace*
- *ReplaceEx*
- *ReplaceEx*
- *Reset*
- *ResyncToTemplate*
- *Test*
- *Upgrade*
- *UploadConfigProperties*
- *Wink*

ClearStatus

Summary	Clears the status information stored in the device. The clear status method causes a LonTalk Clear *Status* network diagnostic message to be sent to the specified device. This clears the error log, last reset cause, and communication counters, which are generally accessed by reading the device's *DetailInfo* object.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**ClearStatus** **Element** **Description** *object* The *AppDevice* object.
Added to API	Prior to LNS Release 3.0.

Commission

Summary	Associates an *AppDevice* object with a physical device, and loads the device's network image. You can also use this method at any time to force a reload of the device's network

	image. Adding an application device to the system is a two-step operation. First, you should define the device and load the device's application image with the *Load* or *LoadEx* methods. Then, you should commission it. You can use the *Add* methods of the *AppDevices* object to define the application device. You can then use the *Commission* method to assign that definition to an actual physical device. If you invoke the *Commission* method while the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**, physical devices will not updated with configuration changes caused by the commission process until the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)**. However, as of LNS 3.20, you can use the *CommissionEx* method to update the physical devices while the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1).** The Neuron ID of the physical device must be set before commissioning it. The Neuron ID of an *AppDevice* is stored in its *NeuronId* property. The commission procedure will also validate that the physical device has the same external interface and program ID as defined for the *AppDevice* object in the OpenLNS database. It will also validate that the physical device is on the channel assigned to the *AppDevice* object in the OpenLNS database. If the physical device is not using the same external interface or program ID as defined in the database, the commission will fail, and either the **NS, #59 lcaErrNsProgramIntfMismatch** or **NS, #38 lcaErrNsProgramidMismatch** exceptions will be thrown. If the physical device is not on the channel assigned to it in the database, the commission will fail, and the **NS, #72 lcaErrNsWrongChannel** exception will be thrown. You should note that this validation will only be performed if the *network management mode* is set to **lcaMgmtModePropagateConfigUpdates (0)**. You can also determine what level of validation will be performed on the device with the *DeviceValidation* property of the *DeviceTemplate* used by the device. When commissioning is complete, an application device is placed in the **lcaStateCnfgOffline** state. Before commissioning an *AppDevice* object, you should make sure that the configuration property information contained in the OpenLNS database for the device is complete, and you should download the configuration property values in the OpenLNS database into the physical device before commissioning the device. You can do so by calling the *DownloadConfigProperties* method on the *AppDevice* object before commissioning. Use the

	lcaConfigPropOptLoadValues and **lcaConfigPropOptLoadUnknown** download options when you call *DownloadConfigProperties*. This will set any unknown configuration properties in the *AppDevice* object in the OpenLNS database to the default values, and then set all configuration property information in the physical device on the network to match the information stored in the OpenLNS database. As a result, once the device has been commissioned, it will contain current values for any configuration properties that have been explicitly set, and default values for any configuration properties that were unknown before the download. Note that by using the *DownloadConfigProperties* method as described above, you will preserve the information stored in the OpenLNS database for the *AppDevice* object, by changing the configuration property information stored in the physical device to match that stored in the OpenLNS database. In some cases, you may want to do the opposite. You may want to change the configuration property information stored in the OpenLNS database for the *AppDevice* object, to match the information stored in the physical device on the network before commissioning. This can usually be accomplished using the *UploadConfigProperties* method, but you cannot use this method until a device has been commissioned. If you are commissioning a device for the first time and want to preserve the configuration property information stored in the physical device, not the information in the database, the solution is to call the *DownloadConfigProperties* and use the **lcaConfigPropOptSetUnknown** value as the download option. All configuration property information in the database will be set to unknown, but the configuration property in the physical device will not be affected. Once the device has been commissioned, you can use the *UploadConfigProperties* method to set the configuration property in the database to match the configuration property information that had been stored in the device. It is recommended that you use the same explicit transaction to call *DownloadConfigProperties* and the *Commission* method. Otherwise, this procedure will take longer, and consume more network bandwidth, than it would when performed within a transaction. Do not read or write to the *State* property of an *AppDevice* or Router in the same explicit transaction with this method. In addition, you must set the *InitialAuthenticationKey* property to the proper value before commissioning an application device that has network management authentication enabled, and has been previously commissioned outside of LNS, or commissioned on a different OpenLNS network. In these circumstances, OpenLNS will not be able to communicate with the device or router without

	knowing its authentication key, since the device has authentication enabled. If the *InitialAuthenticationKey* is set prior to commissioning the device or router, OpenLNS will use this key to authenticate messages sent to the device or router during the commissioning process. See Chapters 5 and 6 of the *OpenLNS Programmer's Guide* for more information on the steps you should take when installing devices on a network and commissioning those devices.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Commission** **Element** — **Description** *object* — The *AppDevice* object to be commissioned.
Added to API	Prior to LNS Release 3.0.

CommussionEx

Summary	Associates an *AppDevice* object with a physical device, and loads the device's network image. This method is the same as the *Commission* method except that it provides additional options that you can choose from when commissioning the device (see the description of the *options* element for more information).
Availability	Local, full, and lightweight clients.
Syntax	*object*.**CommissionEx** *options* **Element** — **Description** *object* — The *AppDevice* object to be commissioned. *options* — A Long value indicating the options to be used when commissioning the device. This determines whether the changes caused by the commission process will be applied to the physical device if the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**. The values for this element, which are stored in the *ConstCommissionFlags* constant, are as follows: **0 lcaCommissionFlagNone** Updates the device as soon as the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0).** Use this option if you do not want the device to be updated if the network management mode property is set to

	lcaMgmtModeDeferConfigUpdates (1). **1 lcaCommissionFlagPropagate** Propagates the device's network image in the OpenLNS database to the physical device when the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1).** If this will cause network inconsistencies, OpenLNS will defer the updates, and the NS, #4039 lcaErrNsUpdatesDeferred exception will be thrown. If the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)**, the device's network image in the OpenLNS database will be propagated to the physical device, regardless of whether this option is set.
Added to API	Prior to LNS Release 3.0.

Decommission

Summary	Sets the *NeuronId* property of the *AppDevice* to "000000000000" (none) and deconfigures the device.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Decommission** **Element** — **Description** *object* — The *AppDevice* object to be decommissioned.
Added to API	Prior to LNS Release 3.0.

Delay

Summary	Represents the average number of milliseconds required for a packet to get onto the channel once queued. This property allows OpenLNS applications to specify the number of milleseconds expected to send a message and receive an acknowledgment on the specified channel, so that automatic timer calculations made by OpenLNS can be affected accordingly. When this property contains the default value of 0, the delay used will be equivalent to the time required for two packet cycles, based on the average packet size and channel transceiver type. When you write to this property, you should note that the delays for transactions on a given channel must be calculated as round-trip delays. Make sure you set this property to a

	value that is based on the amount of time it will take for a request message to be sent on the channel, and for the response message to be sent back on the channel. You should also note that you can set the expected delay for a message to be sent to a specific device by writing to the *Delay* property of the *AppDevice* object.
Availability	Local, full, and lightweight clients.
Syntax	*delayValue* = *channelObject*.**Delay** **Element** **Description** *delayValue* The delay associated with the channel, in milliseconds. The valid range of this property is 0 to 65,535. *Object* *Channel* object to be acted upon.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

DownloadConfigProperties

Summary	Downloads the configuration property values stored in the logical *AppDevice* object into the corresponding physical application device. This method requires that the device being loaded be in a configured state. Furthermore, if you are setting the configuration property values to their default values (for example, the *downloadOptions* element sets the **lcaConfigPropOptSetDefaults** option), this method requires that the configuration property values be uploaded from the device (see the *UploadConfigProperties* method) or imported from the device's external interface file (see the *Import* method).
Availability	Local, full, and lightweight clients.
Syntax	*object*.**DownloadConfigProperties**(*downloadOptions*) **Element** **Description** *object* The *AppDevice* object from which configuratio to be downloaded into a physical device.

	download Options	An Integer value specifying the download options. These values can be ORed together; however, you must set the **lcaConfigPropOptLoadValues (1)** option for the values to be downloaded into the physical device. The values for this element, which are stored in the *ConstConfigPropOptions* constant, are as follows: **0 lcaConfigPropOptLoadDefinitions** This option has no effect during a download. If this is the only option set when you call DownloadConfigProperties, OpenLNS will use the **lcaConfigPropOptLoadValues (1)** as the download option. **1 lcaConfigPropOptLoadValues** Downloads all known values in the *AppDevice* object in the OpenLNS database to the physical device on the network. You must select this value if you want the values to be downloaded into the physical device, regardless of which other flags you set. However, this is the default option used if **lcaConfigPropOptLoadDefinitions (0)** is the only option specified in the call to *DownloadConfigProperties*, or if no flags are specified. **2 lcaConfigPropOptSetDefaults** Sets the configuration property values in the *AppDevice* object to the default configuration property values stored in the DeviceTemplate, and downloads any changed values to the physical device on the network (if the **lcaConfigPropOptLoadValues** flag is set). Manufacturing-only configuration properties are not affected by this option unless it is ORed with **lcaConfigPropOptIncludeMfgOnly (8)**. This option should not be ORed with **lcaConfigPropOptLoadUnknown (4)**. Note that setting SCPTnvType configuration properties to their default values may cause the download operation

		to fail. To exclude those configuration properties from a download, you should OR this value with the **lcaConfigPropOptExcludeNvTypeDefaults (512)** value. **4 lcaConfigPropOptLoadUnknown** Sets all unknown configuration property values in the *AppDevice* object to the default values stored in the DeviceTemplate, and then downloads all known values into the physical device (if the **lcaConfigPropOptLoadValues** flag is set). This will not affect manufacturing-only configuration properties unless ORed with **lcaConfigPropOptIncludeMfgOnly (8)**. This option should not be ORed with **lcaConfigPropOptSetDefaults (2)**, as that would override this option by setting all properties to their defaults. Note that setting SCPTnvType configuration properties to their default values may cause the download operation to fail. To exclude those configuration properties from a download, you should OR this value with the **lcaConfigPropOptExcludeNvTypeDefaults (512)** value. **8 lcaConfigPropOptIncludeMfgOnly** Use this flag to in conjunction with the **lcaConfigPropOptSetDefaults (2)** and **lcaConfigPropOptLoadUnknown (4)** options if you want the download operation to include manufacturing-only configuration properties. Generally, these configuration properties should only be modified during the manufacturing process. However, OpenLNS will not enforce this requirement during a download if this flag is set, since LNS may be the tool used to set the configuration property values during the manufacturing process. **16 lcaConfigPropOptSetUnknown** Sets all configuration properties in the *AppDevice* object in the OpenLNS database to unknown. This has no effect on the values stored in the physical device on

		the network. **32** **lcaConfigPropOptSetMfgOnlyUnknown** Sets all manufacturing only configuration properties to unknown. The values will not be downloaded into the physical device. **64** **lcaConfigPropOptExcludeDeviceSpecific** Downloads only those configuration properties that do not have the device-specific attribute set into the device. For example, if this option is ORed with **lcaConfigPropOptLoadUnknown (4)** and **lcaConfigPropOptLoadValues (1)**, LNS would set all configuration properties that are not device-specific, and whose values are unknown to their default values, and then download those values into the device. This option should not be ORed with **lcaConfigPropOptOnlyDeviceSpecific (128)**. **128** **lcaConfigPropOptOnlyDeviceSpecific** Downloads only device-specific configuration properties into the device. For example, if this option is ORed with **lcaConfigPropOptLoadUnknown (4)** and **lcaConfigPropOptLoadValues (1)**, LNS would set all device-specific configuration properties whose values are unknown to their defaults, and download those values into the device. This option should not be ORed with **lcaConfigPropOptExcludeDeviceSpecific (64).** **256** **lcaConfigPropOptClearUpdatePending** Clears the update pending flag on the device configuration. This value may be used alone, or ORed with other values. If used alone, it will clear the update pending flag of all configuration property values in the device. It may be combined with **lcaConfigPropOptOnlyDeviceSpecific (128)** to clear only the pending flags of device specific values. This value has no effect on the database

	operations initiated by other flags passed to the *DownloadConfigProperties* method. However, updates to the device initiated by the other flags will be cancelled. For example, when combining this value with **lcaConfigPropOptSetDefaults (2)**, the operation will set the values in the database to their defaults, but the pending update flag on the device will be cleared. As a result, those values will not be loaded into the device. **512 lcaConfigPropOptExcludeNvTypeDefault** You can OR this value with either the **lcaConfigPropOptSetDefaults (2)** or **lcaConfigPropOptLoadUnknown (4)** values to prevent setting SCPTnvType configuration property values to their defaults during a download. Setting a SCPTnvType configuration property to its default value may not be allowed due to connection constraints, and therefore would cause the download operation to fail. When ORed with **lcaConfigPropOptSetDefaults (2)**, all configuration properties other than SCPTnvType configuration properties will be set to their default values. When ORed with **lcaConfigPropOptLoadUnknown (4)**, all unknown configuration properties whose values are unknown will be set to their default values, except SCPTnvType configuration properties. This option does not affect the **lcaConfigPropOptLoadValues (1)** value. If the **lcaConfigPropOptLoadValues (1)** value is specified, SCPTnvType configuration properties with known values will be propagated to the device, even if setting their values to the default has been excluded.
Added to API	Prior to LNS Release 3.0.

GetMessagePoint

Summary	Returns a new message point that can be used to send a message to the application device. The returned *MsgMonitorPoint* object can set its *OutputDataPoint* property or *RequestDataPoint* property to

<table>
<tr><td></td><td>send a value on the created monitor point.

The device does not need to be commissioned to call this method.

The device's application determines how it will react to any messages sent using this MsgMonitorPoint object.

Note that you cannot use the message point returned by this method to send a message to a Network Service Device. If you invoke this method on an AppDevice object contained by a NetworkServiceDevice object, the NS, #16 lcaErrNsNotImplemented exception will be thrown.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>msgMpObject = adObject.GetMessagePoint(addressingMode)

Element — Description

msgMpObject — The MsgMonitorPoint object to be returned.

adObject — The AppDevice object to be acted on.

addressingMode — A long value that determines whether Neuron ID or Subnet/Node addressing will be used to send messages to the AppDevice .

The values for this element, which are stored in the ConstAddressingMode constant, are as follows:

0 lcaAddrNeuronId

Neuron ID broadcast addressing will be used. Messages sent using Neuron ID addressing will be sent on all channels in a network, regardless of subnet, and so Neuron ID addressing will add extra traffic to your network. However, unconfigured devices can receive messages sent using Neuron ID addressing. They cannot receive messages sent using Subnet/Node addressing.

1 lcaAddrSubnetNode

Subnet/Node addressing will be used. Generally, you should use Subnet/Node addressing. If you are using a high-performance network interface, you can send messages to multiple devices simultaneously when using Subnet/Node addressing.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.0.</td></tr>
</table>

Load

Summary	Loads the application image specified by the *AppImagePath* property into a device. This method loads the application image specified by the *AppImagePath* property into a device. The application image must be in the binary format used by OpenLNS, (an **APB** file type). A transaction is started implicitly when this method is called. If OpenLNS detects an error while the load is in progress, it automatically cancels the transaction. If the transaction is canceled after the download has begun, the application device will be left in the applicationless state. The state of the device at the end of the application loading process depends on the state of the device before the process began and the image that was loaded, as follows: • If the device was applicationless or unconfigured (for example, did not have a network address) before the loading process began, OpenLNS leaves the device in the unconfigured state. • If the device was configured and the application that was previously in the device and the application that was loaded have the same program ID (and thus the same external interface), OpenLNS restores the device's network image (address and connection information) to the state it was in before the application load started. • If the device was configured, and the application that was previously in the device and the application that was loaded do not have the same program ID (and thus may not have the same external interface), the Object Server leaves the device in the unconfigured state. This will cause the NS, #38 lcaErrNsProgramidMismatch exception to be generated. In this case, you should upgrade the device's interface with the *Upgrade* method before you load the new application. This will ensure that the program IDs of the device and the new application match. • If the device had been previously commissioned, and the device's old application image and its new application image have the same program ID but a different external interface, the OpenLNS Object Server will leave the device in the unconfigured state, and the NS, #59 lcaErrNsProgramIntfMismatch exception will be thrown. Per LonMark guidelines, LNS requires that each program ID be associated with only one external interface. This means that all components and properties of each external interface using a given program ID must be identical. However, LNS may not detect all violations of this rule, as it would be very time consuming to validate this on every commission or after every

	application download. The system image used by the device must be compatible with the application image being loaded for the operation to complete successfully. If it is not, the application image will fail to load and an NS error will be generated. However, some devices support the use of the *LoadEx* method, which will automatically upgrade the system image in the device if it is not compatible with the application image being loaded. If you are re-loading an application image into a device that has already been commissioned, then OpenLNS will reload the values of configuration properties with pending updates into the device after the application download completes. However, it will not reload the values of configuration properties that have been successfully set and loaded into the device. To ensure that the configuration properties are managed as desired and as efficiently as possible, you should clear pending updates on the device before performing the application download, and then re-synchronize the configuration properties in the device and in the OpenLNS database after the application download has completed. To do so, call the *DownloadConfigProperties* method with the *downloadOptions* element set to **lcaConfigPropOptClearUpdatePending (256)** immediately before starting the application download. Once the download is complete, you can preserve the values stored in the OpenLNS database for the device by calling the DownloadConfigProperties method with the *downloadOptions* element set to **lcaConfigPropOptLoadValues (1)** (and optionally ORed with **lcaConfigPropOptLoadUnknown (4)** to set unknown values to their defaults). Or, to preserve the values stored in the device and load them into the OpenLNS database, call the *UploadConfigProperties* method with the *options* element set to **lcaConfigPropOptLoadValues (1)**. The *system management mode* must be set to **lcaMgmtModePropagateConfigUpdates (0)** when you invoke the Load method. If the system management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**, the operation will fail and the NS, #31 lcaErrNsDeferConfigUpdatesMgmntMode exception will be thrown.
Availability	Local, full, and lightweight clients.
Syntax	*appDeviceObject*.**Load** **Element** **Description** *appDeviceObject* The *AppDevice* object to be loaded.
Added to API	Prior to LNS Release 3.0.

LoadEx

Summary	Loads the application image specified by the *AppImagePath* property into a device. This method is the same as the *Load* method except that it provides additional options that you can choose from when commissioning the device. For example, you can specify options to upgrade the system image of the device if it is not compatible with the application image being loaded (see the description of the *options* element for more information). See the *Load* method for more general information on loading application devices.
Availability	Local, full, and lightweight clients.
Syntax	*appDeviceObject*.**LoadEx** *upgradeOption* **Element** — **Description** *appDeviceObject* — The *AppDevice* object to load. *upgradeOption* — A Long value defining the option to be used if the system image stored in the device is incompatible with the application image. The valid values for this element, which are stored in the *ConstLoadOptions* constant, are as follows: **0 lcaLoadOptionsNone** Do not upgrade the system image. In this case, the method works exactly as the *Load* method does. If the device requires a new system image to load the new application in this case, the load will fail and an exception will be generated. **1 lcaLoadOptionsUpgradeSystemImage** OpenLNS supports upgrading the following Neuron models provided that the current firmware version installed in the Neuron chip and the new firmware version to be downloaded are standard system images with versions (and variants) in the listed ranges: • **FT 5000** and **Neuron 5000** (all released versions up through Version 19). Requires at least a 32K flash using a recognized flash driver, or a 32K EEPROM. • **FT 3150** and **3150** (Versions 6–19). Supports both 512 byte EEPROM and 2K EEPROM system image

	variants. Requires the system image to be stored in flash and to have at least 32K bytes of contiguous flash starting at location 0. • **PL 3150** (Versions 14–19). Supports both 512 byte EEPROM and 2K EEPROM system image variants. Requires the system image to be stored in flash and to have at least 32K bytes of contiguous flash starting at location 0. See the *Load* method for more general information on loading application devices. Note that the device will be made applicationless before the system image upgrade is attempted. If the upgrade fails for any reason, an exception will be generated and the device will be left in the applicationless state. The original system image will remain intact. Once the system image has been upgraded successfully, the application loading process begins. The application image must be in the binary format used by OpenLNS, i.e., an APB file type. A transaction is started implicitly when this method is called. If OpenLNS detects an error while the load is in progress, it automatically cancels the transaction. If the transaction is canceled after the download has begun, the application device will be left in the applicationless state.
Added to API	LNS Release 3.2.

MoveEx

Summary	Performs the steps required to move an application device to a new channel. This method is used to perform the steps required to move an application device to a new channel. This method combines the functions performed by the *PreMove* method and *PostMove* methods, and adds a flag parameter to specify advanced options required for a small set of move operations. Applications should use the *PreMove* and *PostMove* methods whenever possible. You must invoke the *MoveEx* method twice during a move.

<table>
<tr><td></td><td>You first need to invoke it with the lcaMovePrePhysical flag (analogous to the PreMove call), and then once the device has been moved, you need to invoke it again with the lcaMovePostPhysical flag.

You should invoke the BeginSession method to begin a session before using the MoveEx method. This will allow your application to avoid some failure scenarios that may occur when devices or routers are moved.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>appDeviceObject.MoveEx newChannelObject, newSubnetObject, flags

Element — Description

appDeviceObject — AppDevice to be moved to a different channel.

newChannelObject — New destination channel for the device.

newSubnetObject — New destination subnet for the device.

flags — A Long value specifying the movement flags. Multiple options may be specified by logically OR'ing individual flag values. The flag values, which are provided in the ConstMoveExFlags constants, are as follows:

1 lcaMovePrePhysical

Specified when invoking MoveEx prior to the physical move

2 lcaMovePostPhysical

Specified when invoking MoveEx following the physical move.

8 lcaMoveOnline

Indicates that moved devices are to be left online (default for routers).

16 lcaMoveOffline

Indicates that moved devices are to be left offline (default for application devices).

32 lcaMoveRestore

Indicates that moved devices will be restored to their original online/offline state. If a device is power cycled or reset as part of the move, this information is lost, and the device will be put online.</td></tr>
<tr><td>Added to API</td><td>Prior to LNS Release 3.0.</td></tr>
</table>

PostMove

Summary	Completes the move of an application device from one channel (or subnet) to another. As part of the move, the device's network address may change; however, the device's *Handle* property cannot change. To move a device from one channel to another, follow these steps: 1. Invoke the *BeginSession* method to begin a session. This will allow your application to avoid some failure scenarios that may occur when devices or routers are moved. 2. Invoke the *PreMove* method. The target channel is validated and the device is deconfigured. However, the device still appears in the database as residing on the original channel. 3. Physically move the device from one channel to another. 4. If the device being moved uses authentication, the *PostMove* method should be called in a different session than the *PreMove* method. This means you should call the *EndSession* method to end the session begun in step 1, and then call the *BeginSession* method to begin a new session. 5. Invoke the *PostMove* method. This method changes the device's channel assignment, does any necessary rebinding, and updates the device's configuration. If you use explicit transactions during this procedure, make sure that there are separate ones for the *PreMove* and *PostMove* steps.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**PostMove** **Element** — **Description** *object* — The *AppDevice* object to be moved from one channel to another.
Added to API	Prior to LNS Release 3.0.

PreMove

Summary	Prepares an application device for movement from one channel (or subnet) to another. When you call this method, you must specify the new channel (*newChannelObject* element). If no new *Subnet* is specified (i.e. the *newSubnetObject* is NULL), then automatic subnet determination is used to determine the new subnet. Note that you can use this method to move a device from one subnet to another, without switching channels. In this case, specify the device's current channel as the *newChannelObject*

	element, and the new subnet as the *newSubnetObject* element. If this method is called while not attached to the network, this message will throw an exception. Ignore it and call the *PostMove* method to move a device while in engineered mode. See the *PostMove* method for more information on the steps required to move an application device.
Availability	Local, full, and lightweight clients.
Syntax	*appDeviceObject*.**PreMove** *newChannelObject, newSubnetObject* **Element** — **Description** *appDeviceObject* — *AppDevice* to be moved from one channel to another. *newChannelObject* — Destination channel for the device. *newSubnetObject* — Destination subnet for the device.
Added to API	Prior to LNS Release 3.0.

PropagateDeviceConfigUpdates

Summary	Propagates any pending configuration changes to the physical device associated with this *AppDevice* object. The *network management mode* can be set to one of two values: **lcaMgmtModePropagateConfigUpdates (0)**, or **lcaMgmtModeDeferConfigUpdates (1)**. When it is set to **lcaMgmtModeDeferConfigUpdates (1)**, all network configuration changes caused by your application will only be applied to the OpenLNS database, and not to the physical devices on the network. Prior to LNS 3.20, these changes would be queued and then applied to all the physical devices as soon as the network management was set back to **lcaMgmtModePropagateConfigUpdates (0)**. However, in LNS 3.20 and beyond, you can use this method to apply device-only configuration changes to a physical device without changing the network management mode back to **lcaMgmtModePropagateConfigUpdates (0)**. This may be useful if you have configuration changes pending for a large number of devices, and only want to apply them to a subset of those devices. The device-only configuration information propagated by this method includes configuration property values and the device location string. It does not include information such as network variable configurations, address table entries and connection information. If you invoke this method and OpenLNS determines that it cannot safely propagate the changes to the physical device, the operation will fail, and the NS, #4039

	lcaErrNsUpdatesDeferred exception will be thrown. In this case, the changes will not be propagated to the device until the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0)**.
Availability	Local, full, and lightweight clients.
Syntax	*device.***PropagateDeviceConfigUpdates** **Element** **Description** *device* The *AppDevice* representing the physical device to which configuration changes are to be propagated.
Added to API	LNS Release 3.20.

Reboot

Summary	Reboots the *AppDevice* . This command should be used with extreme care. Rebooting a device may destroy its communication parameters or otherwise make the device unrecoverable. The result of rebooting a device depends on the firmware state and reboot options specified at the time of manufacture. Rebooting an emulator will cause the emulator to fail. After rebooting a device, it may be necessary to reload the application, and it will be necessary to commission and download configuration properties if applicable.
Availability	Local, full, and lightweight clients.
Syntax	*object.***Reboot** **Element** **Description** *object* The *AppDevice* object to be rebooted.
Added to API	Prior to LNS Release 3.0.

ReleasePendingUpdates

Summary	Releases monitor point update events withheld after the *AcceptIncomingSession* method has been called with the *postponeUpdates* field set to True. When your application receives an uplink session request, you should use the *AcceptIncomingSession* method to accept or reject the request. The method includes a parameter called *doPostponeUpdates*. If this parameter is set to True when a session is accepted, monitor point updates for the network involved in this session will be withheld until the *ReleasePendingUpdates* method is called. This ensures that no monitor point update events that occur before the network that has requested the uplink session is opened are lost, and that the application will receive the monitor point update event that caused the uplink session request.

	If you accept an xDriver session with the *doPostponeUpdates* field set to True, you should open the network involved in the session, and enable all the monitor sets you want to use. Then, call the *ReleasePendingUpdates* method on the *AppDevice* object contained by the network's *MyVni* property. This will release the updates that were withheld. For an example of this, see the example uplink application in the *OpenLDV Programmer's Guide, xDriver Supplement*. Echelon recommends that you only use this method when operating as an Independent client, as using this method in server-dependent mode may disrupt network management operations. If you are operating in server-dependent mode and you invoke this method, an exception will be thrown. However, the monitor point update events will be released. These methods only apply to clients that are using the OpenLDV xDriver to connect to remote network interface (RNI) devices. For an overview of the xDriver, see Chapter 11, *OpenLNS Network Interface Drivers*, of the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*appDevice*.**ReleasePendingUpdates** **Element** **Description** *appDevice* The *AppDevice* object being acted upon.
Added to API	LNS Release 3.06.

Replace

Summary	Replaces one application device with another. This method is typically used to effect repair operations. This method lets you replace one application device with another. The new device receives the same network address and connections as the old device. To automatically load the old device's configuration properties into the new device, use the *ReplaceEx* method. The *ReplaceEx* method performs the same function as the *Replace* method; however, it provides options you can use to determine how the configuration property information on the old device will be managed during the replacement. The new device is specified by setting the *NeuronId* property of the object to the NeuronId of the new device. When the replace method is complete, a new application device is placed in the soft offline state (*lcaStateSoftOffline*) and a new router device is placed in the online state (*lcaCnfgOnline*). You should not use this method on the *AppDevice* property of a *NetworkServiceDevice* because OpenLNS will perform NSD

	replacements automatically in most cases. For information on when you might need to manually replace a Network Service Device and how to accomplish this, see the *PreReplace* method. **Note**: Do not read or write the *State* property of the device in the same explicit transaction as the invocation of this method.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Replace** **Element** **Description** *object* The application device to be replaced.
Added to API	Prior to LNS Release 3.0.

ReplaceEx

Summary	Replaces one application device with another. This method is typically used to effect repair operations. This method is similar to the *Replace* method except that it provides additional options you can use to determine (1) how the configuration property information on the old device will be managed during the replacement, and (2) whether to propagate network configuration changes caused by the replacement if the network management mode (*MgmtMode* property) is set to **lcaMgmtModeDeferConfigUpdates**. See the description of the *options* element for more information. See the *Replace* method for more general information on replacing application devices.
Availability	Local, full, and lightweight clients.
Syntax	o*bject*.**ReplaceEx** *flags* **Element** **Description** *object* The *AppDevice* object to be replaced. *flags* The options which apply to this *Replace* operation. This determines which configuration property values will be passed from the old device to the new device, and whether changes caused by the replacement should be propagated if the network management mode (*MgmtMode* property) is set to **lcaMgmtModeDeferConfigUpdates**. The possible values for this element, which are contained in the *ConstReplaceFlags* constant, are as follows: **0 No options**

		Enter this value if do not want to use any options. **1 lcaReplaceFlagCopy** Copies the values of all configuration properties (CPs) from the old device to the new device. If the old device has no CPs, configuration network variables will be copied, if present. Dynamic configuration network variables will not be copied. This flag may be ORed with **lcaReplaceFlagExcludeMfgOnly (2)** and **lcaReplaceFlagUpload (4)**. **2 lcaReplaceFlagExcludeMfgOnly** Enables the new device to not inherit any manufacturer configuration properties from the old device. This flag must be ORed with **lcaReplaceFlagCopy (1)**. **4 lcaReplaceFlagUpload** Uploads all configuration properties in the old device into the OpenLNS database before the device is removed, and subsequently downloads the CPs into the new device. You can only use this option if your application is still in communication with the old device. This flag must be ORed with **lcaReplaceFlagCopy (1)**. **8 lcaReplaceFlagPropagateUpdates** Propagates the network image contained in the database to the new device, and deconfigures the old device, even if the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)** when you call the *ReplaceEx* method. If any of these changes will cause network inconsistencies, OpenLNS will defer the updates, and the NS, #4039 *lcaErrNsUpdatesDeferred* exception will be thrown. Those updates will not be propagated until the network management mode is set back to **lcaMgmtModePropagateConfigUpdates (0)**. If the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)** when you call the *ReplaceEx* method, the network image contained in the database will be propagated to the new device and the old device will be

	deconfigured, regardless if this option has been set. This flag should be ORed with **lcaReplaceFlagCopy (1)**. If it is not, all configuration property information for the old device will be removed from the OpenLNS database during the replacement, and it will not be stored in the new device.
Added to API	Prior to LNS Release 3.0.

Reset

Summary	Sends a reset command to the application device. The Network Service Device can be reset by invoking this method on the *AppDevice* object owned by the *NetworkServiceDevice* object.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Reset** **Element** **Description** *object* The *AppDevice* object to be reset.
Added to API	Prior to LNS Release 3.0.

ResyncToTemplate

Summary	Updates the *DeviceTemplate* object the application device is using with information from newly modified or accessible resource files. You should call this method on an *AppDevice* object after you have re-imported a device's external interface file with the *Import* method. When this happens, the information stored in the *DeviceTemplate* is updated based on the contents of the new external interface file. As a result, you need to call the *ResyncToTemplate* method to resynchronize the devices using the *DeviceTemplate* with the updated information. You should also call this method on an *AppDevice* after you have updated the *DeviceTemplate* object used by the device with the *ResyncToResources* method. The *ResyncToResources* method updates the *DeviceTemplate* with current information from the resource files. You should call the *ResyncToTemplate* method to re-synchronize the devices using that *DeviceTemplate* with the updated information in the *DeviceTemplate* .
Availability	Local, full, and lightweight clients.

Syntax	*appDevice*.**ResyncToTemplate** *options* **Element** **Description** *appDevice* The *AppDevice* object to be re-synced to reso files. *options* A Long value specifying the options to use when re-synchronizing the device. You can use this element to determine which objects on the device will be re-named during the resynchronization. These options may be ORed together. The possible values for this element, which are contained in the *ConstResyncToTemplateOptionFlags* constant, are as follows: **0 No options** Enter this value if do not want to update any names updated as part of the resynchronization. **1 lcaResyncToTemplateOptionUpdate CpNames** Updates the names of configuration properties on the device. **2 lcaResyncToTemplateOptionUpdate LonMarkObjectNames** Updates the names of LonMarkObject objects on the device. **4 lcaResyncToTemplateOptionUpdate NetworkVariableNames** Updates the names of network variables on the device. **Note**: If you use an illegal value as the *options* element, the LCA, #90 lcaErrVALUeOutOfRange exception will be thrown.
Added to API	LNS Release 3.2.

Test

Summary	Tests an application device. You must set the application device's Neuron ID before invoking this method. You can use this method to verify that an application device is able to communicate on the network, and that a subset of its configuration matches the information contained in the OpenLNS database. To pass the test, a device with the expected Neuron ID must

<table>
<tr><td></td><td>exist on the network, and respond to queries. If the device is installed, it must contain the expected domain/subnet/node address, and respond to subnet/node messages on that address. It must also contain the expected program ID, and the expected network management authentication setting (enabled or disabled). If network management authentication is enabled on the device, it must contain the correct system authentication key. Furthermore, the test verifies that there are no other nodes that respond with the tested nodes subnet/node address.

The results of the test are contained in the AppDevice object's LastTestInfo property.

This service analyzes the result with the assumption that the node's network image is up-to-date; therefore, ensure that the node is up-to-date by invoking this service only while the MgmtMode property is set to lcaMgmtModePropagateConfigUpdates (0). This is because discrepancies between the OpenLNS database and the current configuration of the device on the network are normal, and can be expected while the management mode is lcaMgmtModeDeferConfigUpdates (1). These discrepancies will typically be resolved automatically when the system manage mode has been changed to lcaMgmtModePropagateConfigUpdates (0).

For more information on the Test method, see the Testing Devices and Detecting Device Failures section in Chapter 6 of the OpenLNS Programmer's Guide.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>testStatus = deviceToTest.Test

Element — Description

testStatus — An Integer value with the results returned by the test.

The enumerated values for testStatus, which are contained in the ConstTestResults constant, are as follows:

0 lcaTestResultGood

The device passed all applicable tests.

1 lcaTestResultComm

The OpenLNS Server was unable to communicate with the device using either Neuron ID or its subnet/node addressing. The Network Service Device might not be attached to the network, the Network Service Device might be unconfigured, the target device might not be attached to the network, the target device may be powered off, or the target device may be faulty.</td></tr>
</table>

		2 lcaTestResultCommNeuronId The OpenLNS Server was unable to communicate with the device using Neuron ID addressing. Because the device has not been added or defined, the OpenLNS Server could not test the device using subnet/node ID addressing. The OpenLNS Server might not be attached to the network, the Network Service Device might be unconfigured, the target device might not be attached to the network, the target device may be powered off, or the target device may be faulty. **3 lcaTestResultCommNeuronIdVerified** The OpenLNS Server was able to communicate with the device using subnet/node addressing, but could not communicate with the device using Neuron ID addressing. However, through the use of subnet/node addressed messages, the OpenLNS Server has verified that the device contains the expected Neuron ID. This might be caused by intermittent device or channel failures. You may want to retry this method to see if the error persists. **4 lcaTestResultCommSnode** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using subnet/node addressing. The OpenLNS Server was unable to verify whether the device has been configured with the proper domain/subnet/node address. The device may have reconfigured itself, the device may have been reconfigured by another network management tool, or the device may be faulty. The device may be restored using the *Commission* method. **5 lcaTestResultSnodeVerified** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using subnet/node addressing. However, through the use of Neuron ID addressed messages, The OpenLNS Server has verified that the device contains the expected subnet/node address. This error can occur if the target device is in an unconfigured state. The device may need

		to be recomissioned or it may be in the middle of a two-phase move. It also might be caused by intermittent device or channel failures. You may want to retry this method to see if the error persists. **6 lcaTestResultDuplicateSnode** The OpenLNS Server was able to communicate with the device using Neuron ID addressing. However, when using subnet/node addressing, the responding device contains a different Neuron ID. This failure indicates that multiple devices are configured with the same domain/subnet/node address. This could be caused by attaching a new device to the network which was previously configured as part of another network or by two networks sharing the same media and domain ID. This condition could also occur if the unexpected device was supposed to be removed from the system, but the OpenLNS Server was unable to update the device's network image during a *Remove* method; meanwhile, the removed subnet/node address has been reused for the device being tested. **7 lcaTestResultMismatchDomain** The OpenLNS Server was able to communicate with the device using Neuron ID addressing but could not communicate with the device using subnet/node addressing. Using Neuron ID addressing, the OpenLNS Server found that the domain ID configured in the device does not match the database. This result will be returned if the device's domain address has not be configured or the device has become unconfigured due to a checksum error. **8 lcaTestResultMismatchNeuronId** The OpenLNS Server was unable to communicate with the device using Neuron ID addressing. However, the device that responded to the subnet/node addressed test message contains a different Neuron ID. It appears that the subnet/node address is configured in the responding device but not the tested device. One possible cause is that the tested device is supposed to replace the responding device, but the network images in both devices have not been updated yet.

		You can resolve this problem by physically removing the obsolete device. **9 lcaTestResultMismatchSnode** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using its subnet/node addressing. Through the use of Neuron ID addressed messages, the NSS has found that the subnet/node address configured in the device does not match the database. This result will be returned if the device's domain address has not been configured. **10 lcaTestResultNoNeuronId** The device has not been assigned a Neuron ID. **11 lcaTestResultMismatchProgramId** The OpenLNS Server was able to communicate with the device using both Neuron ID and subnet/node addressing. However, the device does not contain the expected program ID. The device's program ID may have been changed by its application program. Host devices can modify the program ID of their attached network interface. The device should be Removed and Added. **12 lcaTestResultCommSnodeNotVerified** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using subnet/node addressing. The OpenLNS Server did not attempt to verify that the device has been configured with the proper domain/subnet/node address because the device is currently authenticated, and reading the address would result in transmitting the key over the network. The device may have reconfigured itself, the device may have been reconfigured by another network management tool, or the device may be faulty. The device may be restored using the Commission method. **13 lcaTestResultAuthEnabled** The OpenLNS Server was able to communicate with the device using both

		Neuron ID addressing and subnet/node addressing. However, the device has network management authentication *enabled* despite the fact that the device's *AuthenticationEnabled* property is set to FALSE. The device may have enabled network management authentication itself, the device may have been reconfigured by another network management tool, or the device may be faulty. It may be possible to restore the device using the Commission method. **14 lcaTestResultAuthDisabled** The OpenLNS Server was able to communicate with the device using both Neuron ID addressing and subnet/node addressing. However, the device has network management authentication *disabled* despite the fact that the device's *AuthenticationEnabled* property is set to TRUE. The device may have disabled network management authentication itself, the device may have been reconfigured by another network management tool, or the device may be faulty. The device may be restored using the Commission method. **15 lcaTestResultKeyMismatch** The OpenLNS Server was able to communicate with the device using both Neuron ID addressing and subnet/node addressing. The device has network management authentication enabled and the node's AuthentictionEnabled property is set to TRUE. However, the device does not contain the current system authentication key. The device may have changed its authentication key itself, the device may have been reconfigured by another network management tool, or the device may be faulty. It may be possible to restore the device using the Commission method. **16 lcaTestResultInterfaceFailure** The OpenLNS Server was unable to communicate with the OpenLNS network interface. The OpenLNS network interface may have become disconnected or faulty. Exit all OpenLNS applications and perform diagnostics on the OpenLNS network interface using the LONWORKS Interfaces Control Panel application.

	17 lcaTestResultInterfaceNotOnline The OpenLNS network interface that the OpenLNS Server is attempting to use is not Online. Recommission the NetworkServiceDevice of the System object by calling the `System.NetworkServiceDevice.AppDevice.Commission` method, and make sure that the State property of the NetworkServiceDevice object (NetworkServiceDevice.AppDevice.State) is set to lcaOnline. **18 lcaTestResultInterfaceConfigError** The OpenLNS network interface that the OpenLNS Server is attempting to use is not property configured. Recommission the NetworkServiceDevice of the System object by calling the `System.NetworkServiceDevice.AppDevice.Commission` method. *deviceToTest* The application device to be tested.
Added to API	Prior to LNS Release 3.0.

Upgrade

Summary	Makes an application device compatible with an updated external interface without disrupting the existing device configuration and connections. When a device has a new application loaded, or when a device is replaced, the device's external interface may change. You can use the *Upgrade* method to upgrade the device to be compatible with the updated external interface, with minimal disruption. When you call the *Upgrade* method, OpenLNS will upgrade the device's external interface while attempting to preserve the network variable and message tag connections, configuration property settings, and monitor sets on the device. If a device template is not specified in the *dtObject* parameter, the new external interface will be read from the physical device. If you are not going to set the *dtObject* parameter, be sure that the *AppDevice* object's *NeuronId* property is set to the correct value before invoking this method. This will ensure that OpenLNS can communicate with the physical device during the upgrade. If OpenLNS cannot communicate with the physical device, the *Upgrade* method will fail, and the *Result* property of the *UpgradeStatus* object returned by the method will contain the value **lcaUgResNotCommissioned**.

	The *UpgradeStatus* object returned by this method contains information indicating whether or not the upgrade was successful, as well as information describing the changes that were made to each component (for example, LonMark object, network variable, message tag, configuration property, monitor set, and monitor point) of the external interface during the upgrade. You can always access the *UpgradeStatus* object returned the last time the *Upgrade* method was called on an *AppDevice* by reading its *LastUpgradeStatus* property. If the new interface supports dynamic functional blocks and the old one did not, OpenLNS will automatically create dynamic functional blocks and dynamic network variables for any static functional blocks and static network variables in the old interface that do not appear in the new interface. This supports a migration strategy in which a new device will support dynamic functional blocks, instead of defining all of its functional blocks as static. All such added components will have the same programmatic name as the original, unless the original was an array. In this case, an index value will be appended to the name, or if the name does not fit, the name will be truncated. This may lead to the assignment of duplicate names, but an OpenLNS application may change both the user name (*Name*) and programmatic name (*ProgrammaticName*) in this case. A new custom *interface* will be created by OpenLNS to house network variables and LonMarkObjects that are converted from static to dynamic in this fashion. The name used for these custom interfaces is "LcaUpgrade<n>", where <n> is a sequential number reflecting the number of upgrades that the device has gone through since LNS 3.20 was installed. For example, the 3rd time the device is upgraded, this interface would be named "LcaUpgrade3." You can determine which objects have been converted from static to dynamic during the upgrade with this Interface object, or with the *UpgradeInfos* collection. If you need to store these network variables and LonMarkObjects on another interface, you can use the *MoveToInterface* method to do so. This method should be called from within an explicit transaction using the *StartTransaction* method. This allows the upgrade to be easily undone if necessary. For more information on using transactions with LNS, see Chapter 4, *Programming an OpenLNS Application,* of the *OpenLNS Programmer's Guide.* As noted previously, you may need to upgrade a device when you load the device's application. Specifically, if the device was configured, and the application that was previously in the device and the application that was loaded do not have the same program ID (and thus may not have the same external interface), the Object Server will leave the device in the unconfigured state, and you will need to upgrade it.

	Echelon recommends that you upgrade the device before loading the application image. This will allow you to verify that the upgrade completed successfully before you load the application. After a device has been successfully upgraded, and the device application has been loaded (if necessary), you should re-commission it with the *Commission* or *CommissionEx* methods. For more information on replacing devices, see the *Replace* and *ReplaceEx* methods. For more information on loading device applications, see the *Load* and *LoadEx* methods. **Note:** In some cases, it may necessary to upgrade your client's Network Service Device when you change the network interface it is using. Generally, OpenLNS will perform this upgrade automatically, as soon as the system is opened. However, you can prevent LNS from automatically upgrading the Network Service Device by setting the *Flags* property to **lcaFlagsManualNsdUpgrade**. By default, this flag is not set. When the flag is set, you will need to manually perform the upgrade by calling the *Upgrade* method on the *AppDevice* object that represents your client's Network Service Device. There are several factors you will need to consider when doing so. For more information, see the *Network Interfaces and Network Service Devices* section in Chapter 11 of the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*usObject* = *adObject*.**Upgrade** *dtObject, reserved* **Element** — **Description** *usObject* — An *UpgradeStatus* object which contains the status and results of the upgrade. *adObject* — The *AppDevice* object to be upgraded. *dtObject* — Optionally, the *DeviceTemplate* object containing the new external interface. *reserved* — An Integer value reserved for future use. Set to 0.
Added to API	LNS Release 3.0.

UploadConfigProperties

Summary	Uploads all configuration property values from a physical device on the network into the associated *AppDevice* object in the OpenLNS database. A configuration property's value is obtained using the *GetDataPoint* method of the *ConfigProperty* object. The *ConfigProperties* collection containing all the configuration properties on a device is accessed through the *AppDevice* object's *Interface* property (**AppDevice.Interface**). The

	UploadConfigProperties method can be invoked using either the *AppDevice* object or the *Interface* object. You cannot call the *UploadConfigProperties* method on a device until you commission it with the *Commission* method.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**UploadConfigProperties** *options* **Element** **Description** *object* The *AppDevice* object in the OpenLNS database to which configuration properties are to be uploaded. *options* An Integer value specifying the desired upload options. The values for this element, which are stored in the *ConstConfigPropOptions* constant, are as follows: **0 lcaConfigPropOptLoadDefinitions** Reads the template file and loads the configuration property definitions for the device into the OpenLNS database if the configuration property template file on the device has not been imported or uploaded into the OpenLNS database. **1 lcaConfigPropOptLoadValues** Uploads all configuration property values from the physical device on the network to the associated *AppDevice* object in the OpenLNS database. When combined with **lcaConfigPropOptExcludeDeviceSpecific (64)**, configuration properties with the device-specific attribute set will be excluded from the upload. **2 lcaConfigPropOptSetDefaults** Sets the values stored for the device in the OpenLNS database as the default configuration property values in the *AppDevice* object's *DeviceTemplate*. This operation will change the default values that could be applied to any device using the DeviceTemplate. Note that this option will upload values from the physical device, regardless of whether or not it is ORed with **lcaConfigPropOptLoadValues (1)**. In either case, OpenLNS will upload all the configuration property values from the device into the OpenLNS database, and then set all the values in the database as the defaults. You can OR this option with **lcaConfigPropOptLoadUnknown (4)**. In

	this case, only values that are unknown in the OpenLNS database will be uploaded from the physical device. Following that, all the values stored in the OpenLNS database for the device will be set as the defaults in the DeviceTemplate object. This includes the values uploaded by the call to *UploadConfigProperties*, as well as all values that were known in the OpenLNS database before the operation began. **4 lcaConfigPropOptLoadUnknown** This option must be ORed with the **lcaConfigPropOptLoadValues (1)** or the **lcaConfigPropOptSetDefaults (2)** values to have any effect. You can OR this with the **lcaConfigPropOptLoadValues (1)** value to upload all unknown values in the OpenLNS database from the physical device on the network. Alternatively, you can OR this with the **lcaConfigPropOptSetDefaults (2)** values to upload all the unknown values into the OpenLNS database, and then set the uploaded values (as well as all values that were known in the database before the upload) as the device's defaults in the OpenLNS database. Note that all configuration properties in an *AppDevice* object start in the unknown condition. Values that have been explicitly set in the database are not affected by this option. **64 lcaConfigPropOptExcludeDeviceSpecific** Do not upload configuration properties with the *device-specific* attribute set into the OpenLNS database. For example, if this option is ORed with **lcaConfigPropOptLoadValues (1)**, LNS would upload all configuration properties that are not device-specific from the device into the OpenLNS database. This option should not be ORed with lcaConfigPropOptOnlyDeviceSpecific (128). **128 lcaConfigPropOptOnlyDeviceSpecific** Only upload configuration properties with the *device-specific* attribute set into the OpenLNS database. For example, if this option is ORed with **lcaConfigPropOptLoadUValues (1)**, LNS would upload all device-specific configuration properties from the device into the OpenLNS database. This option should not

	be ORed with **lcaConfigPropOptExcludeDeviceSpecific (64)**.
Added to API	Prior to LNS Release 3.0.

Wink

Summary	Invokes the device's Wink task. You can use this method to invoke the device's Wink task. Application devices may be programmed with a Wink task which facilitates the identification and installation of the device. Wink tasks are device specific, but they may include such actions as the flashing of a light, or sounding of an alarm.
Availability	Local, full, and lightweight clients.
Syntax	*appDeviceObject*.**Wink** **Element** **Description** *appDeviceObject* The *AppDevice* object to be winked.
Added to API	Prior to LNS Release 3.0.

Properties

The *AppDevice* object contains the following properties:

- *AliasCapacity*
- *AliasUseCount*
- *AppImagePath*
- *AttachmentStatus*
- *AuthenticationEnabled*
- *BitmapFilePath*
- *Channel*
- *ClassId*
- *CommissionStatus*
- *ConfigurationState*
- *ConnectionUpdateType*
- *Delay*
- *Description*
- *DetailInfo*
- *DeviceTemplate*
- *Extensions*

- *Handle*
- *HasBeenCommissioned*
- *IconFilePath*
- *InitialAuthenticationKey*
- *Interface*
- *Interfaces*
- *LastTestInfo*
- *LastUpgrade Status*
- *Location*
- *LocationInNeuron*
- *MonitorSets*
- *MtHubs*
- *Name*
- *NetworkServiceDevice*
- *NeuronId*
- *NodeId*
- *NonGroupRcvTimer*
- *NsiHandle*
- *NVHubs*
- *Parent*
- *PingClass*
- *PendingNeuronId*
- *Priority*
- *ProgramId*
- *SelfDocumentation*
- *State*
- *SubnetId*
- *Subnets*
- *Subsystems*
- *Upgrade Requirement*

AliasCapacity

Summary	Returns the number of network variable aliases available on the device. This information will be useful to you when managing connections in large systems. For more on network variable aliases, see Chapter 7 of the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*capacity* = *adObject*.**AliasCapacity** **Element** — **Description** *capacity* — The number of network variable selectors available on the device. *adObject* — The *AppDevice* object being acted upon.
Valid Values	The following value is defined for the *Alias* object in the *ConstClassIds* constant. lcaClassIdAliases 58
Data Type	Integer.

Read/Write	Read only.
Added to API	LNS Release 3.20.

AliasUseCount

Summary	Returns the number of network variable aliases currently in use on the device. The *AliasCapacity* property returns the total number of network variable aliases available on the device. This information will be useful to you when managing connections in large systems. For more on network variable aliases, see Chapter 7 of the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*inUse* = *adObject*.**AliasUseCount** **Element** — **Description** *inUse* — The number of network variable aliases currently in use on the device. *adObject* — The *AppDevice* object being acted upon.
Valid Values	The following value is defined for the *Alias* object in the *ConstClassIds* constant. **lcaClassIdAliases** **58**
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

AppImagePath

Summary	Contains the application image path where the application binary image file (.APB extension) associated with the device is stored. Use the *Load* method to load the application image specified by this property.
Availability	Local, full, and lightweight clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.
Comments	If this application device is created from a device template, this property will be set to the same path as the

	DeviceTemplate object's *XifPath* property. If this application device was not created from a device template, this property will be empty and must be assigned by the OpenLNS application.

AttachmentStatus

Summary	Indicates whether the device is attached and in the proper state.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *attachmentStatus* — The attachment status of the device. The values for this element, which are stored in the *ConstDeviceAttachmentStatus* constant, are as follows: **0 lcaDeviceAttached** Indicates that the device that triggered the event is now attached and in the proper state. **1 lcaDeviceInImproperState** Indicates that the device that triggered the event is now attached but not in the proper state. **2 lcaDeviceNotAttached** Indicates that the device that triggered the event is now not attached. *Object* — The *AppDevice* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

AuthenticationEnabled

Summary	Indicates whether an application device uses network management authentication. The following sections describe how to enable and disable authentication. **Enabling Authentication** To enable authentication for any of the devices in a system, follow these steps: 1. Set the system's authentication key using the *System*

<table>
<tr><td></td><td>object's AuthenticationKey property.
2. Set the AuthenticationEnabled property of the AppDevice object that represents the System object's NetworkServiceDevice object to True. This enables authentication for all Network Service Devices on the system. As a result, the authentication key used by each Full and Lightweight client application must match the authentication key established in Step 1 the next time any of those applications opens the network. Note that if you have set the authentication key used by a Full client's network interface to match the authentication key established in step 1, then other clients can use that connection without re-specifying the key, as long the first Full client remains connected to the server.
3. Set the AuthenticationEnabled property to True the AppDevice objects that will use authentication.
Disabling Authentication
To disable authentication for all the devices in a system, follow these steps:
1. Set the AuthenticationEnabled property to False on the AppDevice object that represents the System object's NetworkServiceDevice. This disables authentication for all application devices, routers and Network Service Devices operating on the system.
2. Set the System object's AuthenticationKey property to ffff ffff ffff if the network is using 48-bit authentication, or to ffff ffff ffff ffff ffff if the network is using 96-bit authentication.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>authEnabled = Object.AuthenticationEnabled
Element — Description
authEnabled — Boolean value indicating whether an application device uses network management authentication.
If the SecurityLevel property of the System object is set to lcaSecurityLevelKeyDistributionEnabled, setting this property to True enables network management authentication and installs the system key in the device or router.
If the SecurityLevel is not lcaSecurityLevelKeyDistributionEnabled, setting this property to True implies that network management authentication will be enabled by the Object Server, but the system key will</td></tr>
</table>

		be installed in the device or router side by some external means. Setting this property to **False** results in removing the system key from the device or router side and disabling network management authentication on the device or router side. Only application devices whose **AuthenticationEnabled** property is set to True are permitted to participate in authenticated connections. If an application device participates in authenticated connections, you cannot set this property to **False**.
	Object	The object to be acted on.
Data Type	Boolean.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.
Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** — **Description** *bmpFilePath* — The bitmap path and file name. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).
Added to API	Prior to LNS Release 3.0.

Channel

Summary	Contains the *Channel* object associated with the specified *AppDevice* object. The channel assigned to an *AppDevice* object is determined when you *add the device* to the

	AppDevices collection.
Availability	Local, full, and lightweight clients.
Syntax	*channelObject* = *object*.**Channel** **Element** **Description** *channelObject* The channel object. *object* The *AppDevice* object.
Data Type	Channel object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *AppDevice* object in the *ConstClassIds* constant: **7 lcaClassIdAppDevice** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CommissionStatus

Summary	Stores the commission status of an application device. When an application device is installed using the ad-hoc installation scenario, this property will be set to **lcaCommissionUpdatesCurrent (0)** before and after commissioning. For an application device defined from a *DeviceTemplate* object and channel, this property will be **lcaCommissionUpdatesPending (1)** before

<table>
<tr><td></td><td>commissioning, and lcaCommissionUpdatesCurrent (0) after a successful commission.

This property will be set to lcaCommissionUpdatesPending (1) if any changes have been made to the database that affect the configuration of a router or application device, and those changes have not yet been propagated to the device. Changes may not yet have been propagated to the device because the network management mode is set to lcaMgmtModeDeferConfigUpdates (1), or because the transaction has not yet been committed. If these updates are attempted but fail, this property will be set to lcaCommissionUpdatesFailed (2).

You can force a retry of any failed updates with the RetryUpdates method, or you can set up retry updates at an interval of your choice by setting the UpdateInterval property to a positive value. The CommissionStatus property will be restored to lcaCommissionUpdatesCurrent (0) after the updates have been successfully propagated to the physical devices.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>commStatusValue = object.CommissionStatus

Element — Description

object — The device object to be acted on.

commStatusValue — The commission status of this object. The values for this element, which are stored in the commStatusValue constant, are as follows:

0
lcaCommissionUpdatesCurrent

No outstanding commission updates are pending.

1
lcaCommissionUpdatesPending

Commission updates are currently pending, or in progress. When database changes are made that affect a device's configuration, this value represents the commission status of the device.

The commission status will be changed to lcaCommissionUpdatesCurrent (0) when the changes are successfully propogated to the device, or to lcaCommissionUpdatesFailed (2) if there is a failure to propogate the</td></tr>
</table>

	changes. **2 lcaCommissionUpdatesFailed** Commission updates are currently pending or in progress, and the most recent update attempt failed.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConfigurationState

Summary	Identifies the current configuration state of an application device.
Availability	Local, full, and lightweight clients.
Syntax	*configStateValue* = *appDeviceObject*.**ConfigurationState** **Element** — **Description** *appDeviceObject* — The application device to be acted on. *configStateValue* — The configuration state of this device. The values for this element, which are stored in the *ConstConfigurationState* constant, are as follows: **0 lcaConfigurationNode HasBeenConfigured** The application device has been configured. **1 lcaConfigurationNode HasNeverBeenConfigured** The application device has never been configured. If the *ConfigurationState* property is set to this value, OpenLNS will assume that the address table and network variable configuration entries in the device are set to their unbound values. As a result, it will not update empty address table entries or unbound network variable configuration entries when you commission that device
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConnectionUpdateType

Summary	Specifies in which state the *AppDevice* is placed while performing a connection update. Normally, when connecting network variables, the devices containing the network variables are taken hard-offline prior to making the update. Once all affected devices are in this state, each device is updated, placed in the configured state, and, if they were previously online, will be placed online again. This way the configuration of all devices that are currently online will be consistent at all times. This property allows the device to remain online while these changes are being made. This allows a device to continue sending and receiving updates while its connection status is being changed. This can result in invalid network variable updates being sent or received.
Availability	Local, full, and lightweight clients.
Syntax	*updateType* = *appDeviceObject*.**ConnectionUpdateType** **Element** — **Description** *updateType* — The update type value. The values for this element, which are stored in the *ConstConnectionUpdateTypes* constant, are as follows: **-1 lcaConnectionUpdateTypeNotSet** The device's *ConnectionUpdateType* property has not been set. Its behavior defaults to the value specified using the global *Flags* property. **0 lcaConnectionUpdateTypeOffline** The device will be put offline while connections are taking place. **1 lcaConnectionUpdateTypeOnline** The device will be left online while connections are being made. This setting is not recommended, as it may cause your application to process or send network variable updates using inconsistent network variable configuration. This may lead to misinterpretation of network variable updates on this or other devices. *appDeviceObject* — The application device to be acted on.
Data Type	Integer.

Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Delay

Summary	Represents the average number of milliseconds required for a packet to get sent to the application device once queued. This property allows OpenLNS applications to specify the number of milleseconds expected for an application to send a message and receive an acknowledgment from the device. This enables the automatic timer calculations made by OpenLNS to be adjusted accordingly. This value will be added to any delays calculated by OpenLNS based on the network topology. When this property contains the default value of **0**, OpenLNS will not calculate an extra delay for the device. You can set the expected delay for a message to be sent to a specific channel by writing to the *Delay* property of the *Channel* object.
Availability	Local, full, and lightweight clients.
Syntax	*delayValue* = *Object*.**Delay** **Element** — **Description** *delayValue* — The delay associated with the application device, in milliseconds. The valid range of this property is 0 to 65,535. *Object* — The *AppDevice* object being acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Description

Summary	Stores description information about the *object*.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *AppDevice* object. *object* — The *AppDevice* object to be acted on.
Data Type	String.
Read/Write	Read/write. This property is read-only for *Error* objects.

Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

DetailInfo

Summary	Contains the *DetailInfo* object associated with the specified *AppDevice* object. The *DetailInfo* object contains an error log and communications status information for the *AppDevice* object. When you read this property from an *AppDevice*, OpenLNS will send a query to the device to obtain this information.
Availability	Local, full, and lightweight clients.
Syntax	*detailInfoObject* = *object*.**DetailInfo** **Element** — **Description** *detailInfoObject* — The *DetailInfo* object associated with the application device. *object* — The *AppDevice* object from which to get status information.
Data Type	*DetailInfo* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Device Template

Summary	Contains the *DeviceTemplate* object associated with the specified *AppDevice*.
Availability	Local, full, and lightweight clients.
Syntax	*devTemplateObject* = *appDeviceObject*.**DeviceTemplate** **Element** — **Description** *devTemplateObject* — The device template object. *appDeviceObject* — The *AppDevice* object.
Data Type	*DeviceTemplate* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *AppDevice*. This property returns an Extensions collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique

	identifier set by the manufacturer
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.**Extensions** **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose Extensions collection is being returned.
Data Type	Extensions collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *AppDevice* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property and may also be read using the *Index* property. Some OpenLNS Objects are tracked internally by the OpenLNS Server using a unique handle. Handles may be used with the *ItemByHandle* method as an alternative means of fetching objects.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the object. *object* — The object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

HasBeenCommissioned

Summary	Returns a boolean value indicating whether the application device has been commissioned. This property indicates the commission state of an *AppDevice* object within the OpenLNS database, returning a True value if the *AppDevice* 's *Commission* method has been invoked. If the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**, this property may return True, although the physical application device remains unconfigured on the network. You can use the *CommissionStatus* property to determine

	whether all database changes have been propagated to the physical device.
Availability	Local, full, and lightweight clients.
Syntax	*cmValue* = *appDeviceObject*.**HasBeenCommissioned** **Element** — **Description** *cmValue* — A Boolean type indicating the commission state. *appDeviceObject* — The *AppDevice* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (***.ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** — **Description** *IconFilePathFileName* — Icon file and path name *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

InitialAuthenticationKey

Summary	Contains the initial authentication key to be used when commissioning an application device. This property must be set prior to commissioning an application device that has network management authentication enabled, and has been previously commissioned outside of LNS, or commissioned on a different OpenLNS network. In these circumstances, OpenLNS will not be able to communicate with the device without knowing

	its authentication key because the device has authentication enabled. If the *InitialAuthenticationKey* is set prior to commissioning the device or router, OpenLNS will use this key to authenticate messages sent to the device or router during the commissioning process. This property does not affect the key stored in the device or router after a successful commission. Once the device or router has been commissioned, its authentication key will either be set to FFFFFFFFFFFF or the OpenLNS system authentication key, as determined by the device's *AuthenticationEnabled* property.
Availability	Local, full, and lightweight clients.
Syntax	*authenticationKey* = *object*.**InitialAuthenticationKey** **Element** — **Description** *authenticationKey* — The authentication key to be used when commissioning the application device. *object* — The *AppDevice* object being acted upon.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Interface

Summary	Contains the main *Interface* object associated with the specified *AppDevice* object. This includes the static interface of the device, as well as all custom, virtual interfaces that have been added to the device dynamically, with the *Add* method. The collection of custom interfaces that have been added to a device is contained in the *Interfaces* property.
Availability	Local, full, and lightweight clients.
Syntax	*interfaceObject* = *object*.**Interface** **Element** — **Description** *interfaceObject* — The Interface object retrieved from the object. *object* — The *AppDevice* object to be acted on.
Data Type	Interface object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Interfaces

Summary	Contains the *Interfaces* collection object associated with the specified *AppDevice* object. This is the collection of virtual

	and custom interfaces associated with the device. The Interfaces collection allows virtual and custom interfaces to be added to any device that supports dynamic network variables, dynamic message tags, or dynamic LonMarkObjects. You add custom interfaces to a device with the *Add* method.
Availability	Local, full, and lightweight clients.
Syntax	*interfaceColl = object.***Interfaces** **Element** — **Description** *interfaceColl* — The *Interfaces* collection to be returned. *object* — The *AppDevice* object to be acted on.
Data Type	Interfaces collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LastTestInfo

Summary	Contains the *TestInfo* object containing the results of the last time the *Test* method was called for this device. The *AuxResultData* property indicates which properties of the LastTestInfo object contain useful information.
Availability	Local, full, and lightweight clients.
Syntax	*lastTestInfor* **=** *adObject.***LastTestInfo** **Element** — **Description** *lastTestInfo* — The *TestInfo* object containing the last test results. *adObject* — The *AppDevice* object to be acted on
Data Type	*TestInfo* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LastUpgradeStatus

Summary	Contains the *UpgradeStatus* object returned the last time the *Upgrade* method was called on the *AppDevice* object. This object contains a *Summary* of the changes made during the upgrade of the device's external interface. The *Upgrade* method is used to upgrade a device's external interface. It returns an *UpgradeStatus* object, which contains information indicating whether or not the upgrade was successful, as well as information describing the changes made to each external interface component (for example, LonMark object, network variable, message tag, configuration property,

	monitor set, and monitor point) during the upgrade. If you have not called the *Upgrade* method on a device and attempt to read this property, a **LCA, #6 ObjectNotFound** exception is thrown.
Availability	Local, full, and lightweight clients.
Syntax	*upgradeStatusObject* **=** *adObject*.**LastUpgradeStatus** **Element** — **Description** *upgradeStatusObject* — The *UpgradeStatus* object returned the last time the external interface of the device was upgraded. *adObject* — The *AppDevice* object to be acted on.
Data Type	*UpgradeStatus* object.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Location

Summary	Contains the value of the specified *AppDevice* 's six byte location as a hex string. This property must contain a 12 digit hex string that is a valid hexadecimal value. The only valid characters are 0-9 and A-F. For example, "0000AC43F1B6" is a valid value. The *Location* property allows you to read the *AppDevice* 's location from the OpenLNS database.
Availability	Local, full, and lightweight clients.
Syntax	*locationValue* **=** *object*.**Location** *object*.**Location =** *locationValue* **Element** — **Description** *locationValue* — The location as read from the device as a hex string. *object* — The *AppDevice* object to be acted on.
Data Type	String (6 bytes).
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LocationInNeuron

Summary	Reads the value of the specified *AppDevice* 's six-byte location as a hex string from the physical device's EEPROM.
Availability	Local, full, and lightweight clients.
Syntax	*locationValue* **=** *appDevObject*.**LocationInNeuron**

	appDevObject.**LocationinNeuron =** *locationValue* **Element** — **Description** *locationValue* — The location as read from the device as a hex string. *object* — The *AppDevice* object to be acted on.
Data Type	String (6 bytes).
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MonitorSets

Summary	Contains a collection of all the *MonitorSet* objects on this device. Devices that do not support monitor sets will return an empty collection for this property.
Availability	Local, full, lightweight, and independent clients.
Syntax	*monitorSets* = *adObject*.**MonitorSets** **Element** — **Description** *monitorSets* — The collection of *MonitorSet* objects on this device. *adObject* — The *AppDevice* object to be acted on.
Data Type	*MonitorSets* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

MtHubs

Summary	The message tags in this application device that are hubs in a message tag connection. When a new hub is added, it may not be added to the end of the list of hubs; therefore, a cached copy of the complete hub list should be updated when a new hub is added or deleted.
Availability	Local, full, and lightweight clients.
Syntax	*mtCollection* = *adObject*.**MTHubs** **Element** — **Description** *mtCollection* — The returned *MessageTags* collection. *adObject* — The specified *AppDevice* object.
Data Type	*MessageTags* collection object.
Read/Write	Read only.

Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NetworkServiceDevice

Summary	Returns the parent *NetworkServiceDevice* object of the *AppDevice* object if the application device is associated with an OpenLNS Server or NSI. If the device is an independent application device, reading this property will generate an **lcaNotAnNsiHost** error
Availability	Local, full, and lightweight clients.
Syntax	*nsdObject* = *systemObject*.**NetworkServiceDevice** **Element** — **Description** *systemObject* — The *System* object to be acted on. *nsdObject* — The *NetworkServiceDevice* object.
Data Type	*NetworkServiceDevice* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NeuronId

Summary	Contains the Neuron ID associated with the application device. Neuron IDs are stored as 12-digit hexadecimal strings (for example, "a327ff27ba44").

Availability	Local, full, and lightweight clients.
Syntax	*neuronIdValue* = *object*.**NeuronId** **Element** — **Description** *neuronIdValue* — The NeuronId of the object. *object* — The *AppDevice* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

NodeId

Summary	Contains the node ID associated with an application device. The *NodeId* and *SubnetId* comprise the logical network address assigned to an *AppDevice* when the *Add* method is invoked on it. Each *AppDevice* is allocated a single node ID. However, you should note that Network Service Devices are allocated two Node IDs. **Note**: As of OpenLNS, you can write to this property.
Availability	Local, full, and lightweight clients.
Syntax	*nodeIdValue* = *object*.**NodeId** **Element** — **Description** *nodeIdValue* — The NodeId of the object. *object* — The *AppDevice* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

NonGroupRcvTimerNonGroupRcvTimer_Property

Summary	The value of the device's non-group receive timer. This determines the time period within which receiving devices will treat messages from the same source with the same reference ID, but from a different connection, as duplicate messages. The default is calculated based on the network topology, channel delays and current connections that the device is involved in. The value is encoded as follows: **Encoded Value** — **Seconds** 0 — 0.128 1 — 0.192 2 — 0.256 3 — 0.384 4 — 0.512 5 — 0.768 6 — 1.024 7 — 1.536 8 — 2.048 9 — 3.072 10 — 4.096 11 — 6.144 12 — 8.192 13 — 12.288 14 — 16.384 15 — 24.576
Availability	Local, full, and lightweight clients.
Syntax	*timerValue* = *adObject*.**NonGroupRcvTimer** **Element** — **Description** *timerValue* — The non-group receive timer value. *adObject* — The *AppDevice* to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

NsiHandle

Summary	Returns the unique identifier assigned to an NSI by the OpenLNS Server. This handle is different than the *NodeHandle* and *Handle* properties. For the *AppDevice* object, this property returns the handle for the device's parent *NetworkServiceDevice* object, provided the device is associated with an NSI or OpenLNS Server. If the device is an independent application device, the return value will be zero.
Availability	Local, full, and lightweight clients.

Syntax	*handleValue* = *nsdObject*.**NsiHandle** **Element** **Description** *handleValue* The NSI's handle. *nsdObject* The *NetworkServiceDevice* to be acted on
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NVHubs

Summary	The network variables in the connections of this application device that represent the hubs of those connections (note that the hubs are not necessarily on this application device). When a new hub is added, it will not neccessarily be added to the end of the list of hubs; therefore a cached copy of the complete hub list should be updated when a new hub is added or deleted.
Availability	Local, full, and lightweight clients.
Syntax	*nvColl* = *adObject*.**NVHubs** **Element** **Description** *adObject* The *AppDevice* object to be acted on. *nvColl* The collection of *NetworkVariables* to be returned.
Data Type	*NetworkVariables* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy. For example, an *AppDevice* object's parent can be a *NetworkServiceDevice* object or a *AppDevices* collection object
Availability	Local, full, lightweight, and independent clients. Note that not all objects that contain this property are available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*.

	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.	

PingClass

Summary	Determines the frequency with which a device is tested (pinged) to determine if it is still attached to the network. This property classifies devices based on the probability that the device may be detached. The higher the probability, the more frequently the device will be pinged. The Object Server assumes a device to be detached if it cannot communicate with that device three consecutive times.	
Availability	Local, full, and lightweight clients.	
Syntax	*pingClassValue* = *Object*.**PingClass**	
	Element	**Description**
	Object	The device object to be acted on.
	pingClassValue	The ping class of this object. The valid values for this element, which are provided in the *ConstPingClass* constant, are as follows: **0 lcaPingClassDefault** If this value is written to the PingClass property, OpenLNS will use the default **lcaPingClassStationary (3)** value. **1 lcaPingClassMobile** Class for nodes which move frequently. **2 lcaPingClassTemporary** Class for temporary nodes. **3 lcaPingClassStationary** Class for nodes which rarely move. This is the default value. **4 lcaPingClassPermanent** Class for nodes which never move. **Note**: You change the ping interval that applies to each class with the

	System object's *PingIntervals* property.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

PendingNeuronIdPendingNeuronId_Property

Summary	Contains the neuron ID that will be set when the device is commissioned. If a device is commissioned with one Neuron ID, then replaced, then the new Neuron ID is set, the *NeuronId* property will still show the old Neuron ID until the *Commission* method is called. This property allows the new Neuron ID to be read before commissioning.
Availability	Local, full, and lightweight clients.
Syntax	*neuronId = adObject.***PendingNeuronId** **Element** — **Description** *neuronId* — The pending neuron ID. *adObject* — The *AppDevice* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Priority

Summary	Determines the device's priority assignment on its channel. An assignment of **0** indicates that the object is not assigned a priority slot. If the object is not assigned a priority slot, messages with priority will still use priority buffers, and will still be sent before messages without priority. In addition, messages with priority have the priority bit set, so that routers will send them out using priority buffers. If a router has a priority slot defined, the message will be forwarded on that slot.
Availability	Local, full, and lightweight clients.
Syntax	*priorityValue* = o*bject.***Priority** **Element** — **Description** *priorityValue* — The priority value assigned to the object. The enumerated values that you can set for this property, which are stored in the *ConstLNSNodePriority* constant, are as follows:

		127 lcaLNSNodePriorityMax Represents the maximum number of priority slots on any LonTalk channel. Do not set the MaxPriority property to a value greater than this. **255 lcaLNSNodePriorityAny** The Object Server will assign the *AppDevice* object the next available, or least used, priority slot on the channel.
	object	The *AppDevice* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

ProgramId

Summary	Stores the program ID for the application device. Every LonMark compliant LONWORKS application device uses a unique, 16 digit, hexadecimal standard program ID that uses the following format: FM:MM:MM:CC:CC:UU:TT:NN See the *Devices Interfaces* section in Chapter 6 of the *OpenLNS Programmer's Guide* for a description of the format used to display program IDs.	
Availability	Local, full, and lightweight clients.	
Syntax	*programIdValue* = *object*.**ProgramId**	
	Element	**Description**
	programIdValue	The program ID value of the object.
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

SelfDocumentation

Summary	Stores the self-documentation string of the application device. The length of the string is not provided as a separate property. To get the length, get the *descriptionString*, and calculate the length from it. Note that this property returns only the user portion (which follows the LonMark portion, if any) of the self-documentation string

Availability	Local, full, and lightweight clients.
Syntax	*descriptionString* = *object*.**SelfDocumentation** **Element** — **Description** *programIdValue* — The program ID value of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

State

Summary	Describes the state of a device. You cannot set the state of a device in the same transaction as an *Add*, *Commission*, or *Replace* method. You cannot set the state of a device during the definition phase of the predefined components installation scenario. The state change may fail if the device has not been completely updated due to a previous communication problem or if the device has not been commissioned. If a failure occurs, use the *Commission* method to force an update of the device. Devices cannot receive network events related to monitor and control while they are set to an offline state. For example, if the State property of an app device installed on the network is set to offline, then that device will not be able to receive incoming network variable events. While considering this, it is important to note that devices are placed offline while they are being configured or commissioned. For example, if you add or remove a connection between an application device and the Network Service Device, both the application device and Network Service Device will be taken offline while the change is being made.
Availability	Local, full, and lightweight clients.
Syntax	*stateValue* = *object*.**State** **Element** — **Description** *stateValue* — The state of the device (online or offline). The enumerated values for this property, which are stored in the *ConstDeviceStates* constant, are as follows: **Note:** The only two values that may be written to this property are **lcaStateCnfgOnline**(4) and **lcaStateSoftOffline**(12). All other

		properties are read-only. **2 lcaStateUncnfg** The application is loaded but the configuration is either not loaded, being reloaded, or deemed corrupted due to a configuration checksum error. A Neuron Chip also can make itself unconfigured by calling the Neuron C function **go_unconfigured()**. The device's service LED flashes at a one second rate in this state. **3 lcaStateNoApplUncnfg** No application is loaded yet, the application is in the process of being loaded, or the application has been deemed corrupted due to an application checksum error or signature inconsistency. The application does not run in this state. The device's service LED is steadily on in this state. **4 lcaStateCnfgOnline** Normal device state. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The device's service LED is off in this state. **6 lcaStateCnfgOffline** Application loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The device's service LED is off in this state. **12 lcaStateSoftOffline** The device has an application, is configured, and is soft-offline. It will go online when it is reset or when requested to go online. The device's service LED is off in this state. **140 lcaStateCnfgBypass** The application confirmed the offline request, but is still running (bypass

	mode). The device's service LED is off in this state. *object* The device to be acted on.
Data Type	Integer.
Read/Write	Read /write.
Added to API	Prior to LNS Release 3.0.

SubnetId

Summary	Contains the ID of the subnet. The *AppDevice* object's *SubnetId* property identifies the subnet the device is part of. This property can be used in conjunction with the *NodeId* property to uniquely identify a device. **Note**: As of OpenLNS, you can write to this property.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**SubnetId** **Element** **Description** *returnValue* The subnet ID of the device. *object* The *AppDevice* to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Subnets

Summary	Contains the *Subnets* collection object associated with the specified *AppDevice*. The *Subnets* collection contains subnets that are associated with *object*. For example, the *AppDevice* objects' *Subnets* property returns a *Subnets* collection object corresponding to the subnet for the device's index 0 domain entry.
Availability	Local, full, and lightweight clients.
Syntax	*subnetCollection* = *object*.**Subnets** **Element** **Description** *subnetCollection* The returned subnet collection *object* The *AppDevice* object to be acted upon.
Data Type	*Subnets* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Subsystems

Summary	Contains the *Subsystem* collection object associated with the specified *AppDevice*. A *Subsystem* object can in turn contain a collection of Subsystems. A *System* object's *Subsystems* collection contains two default Subsystems upon creation. These Subsystems are named "ALL", which lists all of the devices in the system and "Discovered", which lists all devices discovered by the object server that have not yet been associated with a subsystem. This includes both unconfigured devices discovered by the NSS and configured devices that were added by some other network management application that does not use the Object Server. Subsystems collection objects accessed through *AppDevice* objects represent the Subsystems that contain the specified *AppDevice*.
Availability	Local, full, and lightweight clients.
Syntax	*subsystemCollection* = *object*.**Subsystems** **Element** — **Description** *subsystemCollection* — Subsystems collection associated with the *object*. *object* — The *AppDevice* object.
Data Type	Subsystems collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

UpgradeRequirement

Summary	Indicates whether the application in the device matches the application information contained in the OpenLNS database. If a new application is loaded into a device using the *Load* method, the program ID of the application will not match the *ProgramId* property of the *AppDevice* object. When this occurs, the device's external interface can be upgraded with minimal disturbance to existing connections using the *AppDevice* object's *Upgrade* method.
Availability	Local, full, and lightweight clients.

Syntax	*upgradeReqValue* = *adObject*.**UpgradeRequirement** **Element** — **Description** *upgradeReqValue* — An enumerated value indicating whether the device's application matches the application information in the OpenLNS database. The values for this property, which are stored in the *ConstUpgrade Requirement* constant, are as follows: **0 lcaUgReqNotRequired** Indicates that the device's application does not need to be upgraded. **1 lcaUgReqProgramIdMismatch** Indicates that the program ID of the application does not match the *ProgramId* property of the *AppDevice* object. *adObject* — The *AppDevice* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

AppDevices

An *AppDevices* object is a collection of *AppDevice* objects. An instance of this collection is typically used to hold all of the application devices in a subsystem. The following table summarizes the *AppDevices* object.

Description	Represents a collection of *AppDevice* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Subsystem* object.
Default Property	*Item* property.
Methods	• *Add* • *AddReference* • *ItemByHandle* • *ItemByNeuronID* • *Remove*
Properties	• *ClassID* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *AppDevices* object contains the following methods.

- *Add*
- *AddReference*
- *ItemByHandle*
- *ItemByNeuronID*
- *Remove*

Add

Summary	Defines a new *AppDevice* object. This method adds a device to the system, but does not install the physical device. The *Commission* method is used to install the physical device after that device is defined using this method. Adding an application device to the system is a two or three step operation: definition, loading (optional), and commissioning. 1. The *Add* method is used to define the device. 2. The *Load* method is optionally used to write the application image on the device. 3. The *Commission* method is used to write the network image on the physical device, including its network address. The *Add* method can be used whether OpenLNS is physically attached to the network or not, and while the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0)** or **lcaMgmtModeDeferConfigUpdates (1)**. You can specify the device's definition when you call the *Add* method by supplying a *DeviceTemplate* object as the *deviceTemplateObject* parameter. If a *DeviceTemplate* is provided, then the device definition is taken from there. You can initialize a DeviceTemplate by importing an external interface file (**.XIF** and **.XFB** extensions) with the *Import* method. For more information on device interfaces, see Chapter 6 of the *OpenLNS Programmer's Guide*. If you do not specify the *DeviceTemplate* object, the device template will be assigned (and recovered from the device if necessary) when the device is commissioned. In this case, the device's interface will be unavailable until the device has been commissioned. You should specify the device template whenever possible. The *Add* method (in combination with the *Commission* method when either the *deviceTemplateObject* or *channelObject* element is omitted) causes the following to occur: • The device is assigned a subnet/node address in the system's domain.

<table>
<tr><td></td><td>

The device's network variables and message tags are unbound;
The device's priority slot is set to zero.
The device's state is set to lcaStateCnfgOffline.
The device's use of network management authentication is disabled.
The device's non-group receive timer to be set to 2,048 ms. Other parts of the network image are not modified.

You cannot add devices to the AppDevices collection objects contained in either the ALL or Discovered subsystems, as write access to the app devices contained within these pre-defined subsystems is limited. Devices may not be added to, or deleted from, these subsystems. However, you can create additional subsystems for your app devices with the Add method of the Subsystems collection object.

Note: Do not read or write the State property of the AppDevice in the same explicit transaction in which it is added or commissioned.
</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>
appDeviceObject = appDevicesColl.Add(deviceName, deviceTemplateObject, channelObject, subnetObject)
<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>appDeviceObject</td><td>The newly defined AppDevice object.</td></tr>
<tr><td>appDevicesColl</td><td>The AppDevices collection object.</td></tr>
<tr><td>deviceName</td><td>A String containing the name of the application device.</td></tr>
<tr><td>deviceTemplateObject</td><td>DeviceTemplate object to be associated with the device.

If you specify the deviceTemplateObject parameter, then you can also provide a previously defined Channel object as the channelObject parameter to explicitly assign the device's channel.

A channelObject should always be provided for multi-channel networks with LonWorks routers configured as repeaters.

If no channel is specified when the device is created, the OpenLNS Object Server will use automatic channel determination to assign the channel when the device is</td></tr>
</table>
</td></tr>
</table>

		commissioned. You should specify the channel whenever possible.
	channelObject	The *Channel* object of the channel in which the device is to be placed.
	subnetObject	The *Subnet* object of the subnet to which the device is to be assigned. This parameter is optional, and is not typically specified. If not specified, the Object Server determines the subnet assignment, creating one if necessary. To explicitly assign the subnet, specify a previously defined *subnetObject*. This allows you to take advantage of subnet broadcast messaging. Note that you can use the *MoveEx* method to move a device to a different subnet after the device has been added.
Added to API	Prior to LNS Release 3.0.	

AddReference

Summary	Adds an *AppDevice* object reference to an *AppDevices* collection. This method can be used to add an existing *AppDevice* to multiple *Subsystem* objects (by invoking this method on the *AppDevices* collection in those *Subsystem* objects). For example, an application might contain both a logical hierarchy for the system (where each subsystem represents a function, such as lighting control or the first stage of a batch process) and a physical hierarchy for the system (where each subsystem represents a physical place such as a room or cell). This method allows *AppDevices* to be placed within both hierarchies in the appropriate subsystems. When initially defining a device, it is first added to a single subsystem. References to the device may then be added to other subsystems. The device is not deleted from the OpenLNS database or decommissioned until all references have been deleted. The device's association with the first subsystem is also treated as a reference, so it may be removed from its initial subsystem at any time. If *AddReference* is used to "add" a device discovered in one of the Discovered.<xxx> subsystems, invoking this method will cause the device to be removed from its original discovered subsystem.
Availability	Local, full, and lightweight clients.
Syntax	*collection*.**AddReference** *object*

	Element	**Description**
	collection	The *AppDevices* collection to gain the reference.
	object	The *AppDevice* object to be added.
Added to API	Prior to LNS Release 3.0.	

ItemByHandle

Summary	Retrieves an *AppDevice* object, specified by its handle, from an *AppDevices* collection. The *AppDevice* object to be retrieved must be specified by its handle.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.**ItemByHandle**(*handle*)
	Element **Description**
	object The object retrieved from the collection.
	coll The collection object.
	handle A Long value specifying the handle of the object to retrieve.
Added to API	LNS Release 3.0.

ItemByNeuronID

Summary	Retrieves an *AppDevice* object, specified by its *NeuronId* property, from an *AppDevices* collection.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.**ItemByNeuronId**(*neuronId*)
	Element **Description**
	object The *AppDevice* retrieved from the collection.
	coll The *AppDevices* collection object.
	neuronId A String specifying the Neuron ID of the *AppDevice* object to be retrieved.
Added to API	LNS Release 3.0.

Remove

Summary	Removes an application device from a subsystem. If the application device is not a member of any other subsystems, then it is removed from the system and the network. An application device is removed from a subsystem by removing the corresponding *AppDevice* object from the *AppDevices* collection object owned by that *Subsystem* object. If the application device is in any other subsystems, then nothing further is done.

	If the application device in not a member of any other subsystem, then the application device is completely removed from the system. The device is removed from all connections, removed from the system domain, and placed in the unconfigured state; the device's channel ID is set to 0. No other changes are made in the device's network image. The NSS *Handle* is also freed and is available for use by the Object Server. An *AppDevice* object can only be removed from a "regular" subsystem. The *AppDevices* collection object's *Remove* method cannot be used on the All, Discovered.Installed, or Discovered.Uninstalled subsystems.
Availability	Local, full, and lightweight clients.
Syntax	*appDevicesColl*.**Remove** *indexName* **Element** — **Description** *appDevicesColl* — The *AppDevice* collection object containing the device to be removed. *indexName* — A Long value specifying the collection index of the *AppDevice* object to remove, or a String value specifying the name of the *AppDevice* object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *Alias* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *AppDevices* object in the *ConstClassIds* constant: **8 lcaClassIdAppDevices**

	object The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an object from a collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** **Description** *retrievedObject* The object retrieved from the collection. *collObject* The collection object to be acted on. *index* A Long type specifying the ordinal index of the object to retrieve.

	stringExpression A string type specifying the name of the object to retrieve.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.

Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

Application

Description	Used to store application version, name, and state information.
Added to API	LNS Release 3.0.
Accessed Through	*NetworkServiceDevice* object.
Default Property	*Item* property.
Methods	None.
Properties	• *ClassId* • *MajorApiVersion* • *MajorVersion* • *MinorApiVersion* • *MinorVersion* • *Name* • *Parent* • *State*

Methods

The *Application* object does not contain any methods.

Properties

The *Application* object contains the following properties.

- *ClassId*
- *MajorApiVersion*
- *MajorVersion*
- *MinorApiVersion*
- *MinorVersion*
- *Name*
- *Parent*
- *State*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Application* object in the *ConstClassIds* constant: **70 lcaClassIdApplication** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

MajorAPIVersion

Summary	Contains the major version of LNS used when the application was compiled. This property is used with the *MinorApiVersion* property to provide the full version information. This property is set to 255 if there is no available version information.
Availability	Local, full, and lightweight clients.
Syntax	*majApiVersion* = *appObject*.**MajorApiVersion** **Element** — **Description** *majApiVersion* — The major version of OpenLNS used when this application was compiled. *appObject* — The *Application* object to be acted on.
Added to API	LNS Release 3.0.

MajorVersion

Summary	Contains the major version of the application. This property is used with the *MinorVersion* property to provide the full version information. This property is set to 255 if there is no available version information.
Availability	Local, full, and lightweight clients.

Syntax	*majVersion* = *appObject*.**MajorVersion** **Element** **Description** *majVersion* The major version number of this application when it was compiled. *appObject* The Application object to be acted on.
Added to API	LNS Release 3.0.

MinorAPIVersion

Summary	Contains the minor version of LNS used when the application was compiled. This property is used with the *MajorApiVersion* property to provide the full version information. This property is set to 255 if there is no available version information.
Availability	Local, full, and lightweight clients.
Syntax	*minApiVersion* = *appObject*.**MinorApiVersion** **Element** **Description** *minApiVersion* The minor version of LNS used when this application was compiled. *appObject* The Application object to be acted on.
Added to API	LNS Release 3.0.

MinorVersion

Summary	Contains the minor version of the application. This property is used with the *MajorVersion* property to provide the full version information. This property is set to 255 if there is no available version information.
Availability	Local, full, and lightweight clients.
Syntax	*minVersion* = *appObject*.**MinorVersion** **Element** **Description** *minVersion* The minor version number of this application when it was compiled. *appObject* The *Application* object to be acted on.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. For *AppDevice* objects, this property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.

Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy. For example, an *AppDevice* object's parent can be a *NetworkServiceDevice* object or a *AppDevices* collection object
Availability	Local, full, lightweight, and independent clients. Note that not all objects that contain this property are available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

State

Summary	Describes the state of the application.
Availability	Local, full, and lightweight clients.
Syntax	*stateValue* = *appObject*.**State** **Element** — **Description** *stateValue* — The state of the device (online or offline). The enumerated values for this property, which are stored in the

		ConstApplicationState constant, are as follows: **0 LcaApplOnline** The application is currently executing. **1 LcaApplOffline** The application is not currently executing.
	appObject	The *Application* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

BufferConfiguration

A *BufferConfiguration* object contains the communication buffer parameters for a *RouterSide* object. Changes made to the buffer count and size properties are temporary. To save changes to these properties and propagate the new values on the network, the *BufferConfiguration* property of the *RouterSide* object must be set with the modified *BufferConfiguration* object.

The maximum buffer size for a router side is determined when the router's external interface file is imported. No changes may be made to the *BufferConfiguration* property of the *RouterSide* object that would cause the total memory usage value to be exceeded. You can specify a different external interface file that allows for greater memory for a router if you want to change this value. The external interface file for a router must be specified before the router is commissioned; therefore, you must follow this procedure to do so:

1. Use the *Decommission* method on the router that needs to be modified.
2. Use the *AddEx* method to add a replacement router object to the OpenLNS database. Use **lcaRouterFlagNoSplit (0)** as the *flags* option when you call the method.
3. Import the external interface file that will be used by the router. Note that the LonWorks\Import\Router.xif is the correct external interface file to use for all LonPoint routers, and for any routers based on the RTR-10 SIM Module.
4. Set the *ProgramId* property of each *RouterSide* object of the replacement router to match that of the external interface file used by that type of router. This should be the external interface file imported in Step 3.
5. Use the *Commission* method on the replacement router. Once you have commissioned the router, you can set the properties of the *BufferConfiguration* object to use the additional memory allowed by the new external interface file.
6. *Remove* the old router.

The following table summarizes the *BufferConfiguration* object.

Description	Contains the communication buffer parameters for a *RouterSide* object.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*RouterSide* object.

Default Property	None.
Methods	None.
Properties	• *ClassId* • *InputBufferCount* • *InputBufferSize* • *OutputBufferCount* • *OutputBufferSize* • *PriorityBufferCount* • *PriorityBufferSize*

Methods

The *BufferConfiguration* object does not contain any methods.

Properties

The *BufferConfiguration* object contains the following properties:

- *ClassId*
- *InputBufferCount*
- *InputBufferSize*
- *OutputBufferCount*
- *OutputBufferSize*
- *PriorityBufferCount*
- *PriorityBufferSize*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *BufferConfiguration* object in the *ConstClassIds* constant: **56 lcaClassIdBufferConfiguration** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

InputBufferCount

Summary	Contains the value of the input buffer count used for router configuration. Changes made to this property are temporary. To save the new value and send it to the *Router* on the network, set the *BufferConfiguration* property of the *RouterSide* object to point to the modified *BufferConfiguration* object. The valid count values are discrete. If a specified value is invalid, the buffer count property will be rounded up to the next allowed value. Also, the combination of all input, output, and priority buffers and counts must not exceed the available memory. See the *LONWORKS Router User's Guide* for more information.
Availability	Local, full, and lightweight clients.
Syntax	*bufCount* = *bcObject*.**InputBufferCount** **Element** — **Description** *bufCount* — The input buffer count. *bcObject* — The *BufferConfiguration* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

InputBufferSize

Summary	Contains the value of the input buffer size used for router configuration. Changes made to this property are temporary. To save the new value and send it to the Router on the network, set the *BufferConfiguration* property of the original *RouterSide* object to point to the modified *BufferConfiguration* object. The valid size values are discrete. If a specified value is invalid, the buffer size property will be rounded up to the next allowed value. Also, the combination of all input, output, and priority buffers and counts must not exceed the available memory. See the *LONWORKS Router User's Guide* for more information.
Availability	Local, full, and lightweight clients.
Syntax	*bufSize* = *bcObject*.**InputBufferCount** **Element** — **Description** *bufSize* — The input buffer size. *bcObject* — The *BufferConfiguration* object to be acted on.

Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

OutputBufferCount

Summary	Contains the value of the output buffer count used for router configuration. Changes made to this property are temporary. To save the new value and send it to the Router on the network, set the *BufferConfiguration* property of the original *RouterSide* object to point to the modified *BufferConfiguration* object. The valid count values are discrete. If a specified value is invalid, the buffer count property will be rounded up to the next allowed value. Also, the combination of all output, output, and priority buffers and counts must not exceed the available memory. See the *LONWORKS Router User's Guide* for more information.
Availability	Local, full, and lightweight clients.
Syntax	*bufCount* = *bcObject*.**OutputBufferCount** **Element** — **Description** *bufCount* — The output buffer count. *bcObject* — The *BufferConfiguration* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

OutputBufferSize

Summary	Contains the value of the output buffer size used for router configuration. Changes made to this property are temporary. To save the new value and send it to the Router on the network, set the *BufferConfiguration* property of the original *RouterSide* object to point to the modified *BufferConfiguration* object. The valid size values are discrete. If a specified value is invalid, the buffer size property will be rounded up to the next allowed value. Also, the combination of all output, output, and priority buffers and counts must not exceed the available memory. See the *LONWORKS Router User's Guide* for more information.
Availability	Local, full, and lightweight clients.
Syntax	*bufSize* = *bcObject*.**OutputBufferCount**

	Element **Description** *bufSize* The output buffer size. *bcObject* The *BufferConfiguration* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

PriorityBufferCount

Summary	Contains the value of the priority buffer count used for router configuration. Changes made to this property are temporary. To save the new value and send it to the Router on the network, set the *BufferConfiguration* property of the original *RouterSide* object to point to the modified *BufferConfiguration* object. The valid count values are discrete. If a specified value is invalid, the buffer count property will be rounded up to the next allowed value. Also, the combination of all priority, priority, and priority buffers and counts must not exceed the available memory. See the *LONWORKS Router User's Guide* for more information.
Availability	Local, full, and lightweight clients.
Syntax	*bufCount* = *bcObject*.**PriorityBufferCount** **Element** **Description** *bufCount* The priority buffer count. *bcObject* The *BufferConfiguration* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

PriorityBufferSize

Summary	Contains the value of the priority buffer size used for router configuration. Changes made to this property are temporary. To save the new value and send it to the Router on the network, set the *BufferConfiguration* property of the original *RouterSide* object to point to the modified *BufferConfiguration* object. The valid size values are discrete. If a specified value is invalid, the buffer size property will be rounded up to the next allowed value. Also, the combination of all priority, priority, and priority buffers and counts must not exceed the available memory. See the *LONWORKS Router User's Guide*

	for more information.
Availability	Local, full, and lightweight clients.
Syntax	*bufSize* = *bcObject*.**PriorityBufferCount** **Element** **Description** *bufSize* The priority buffer size. *bcObject* The *BufferConfiguration* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Channel

A *Channel* object represents a single instance of a LONWORKS channel. Channel objects contain communication and transceiver information common to all devices on the channel. The following table summarizes the *Channel* object.

Description	Represents a single instance of a LONWORKS channel. Channel objects contain communication and transceiver information common to all devices on the channel.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object *Channels* collection object *RouterSide* object
Default Property	*Name*.
Methods	None.
Properties	• *AltPathType* • *AppDevices* • *BitmapFilePath* • *ClassId* • *Delay* • *Description* • *Extensions* • *Handle* • *IconFilePath* • *MaxPriority* • *Name* • *Parent* • *RouterDevices* • *TransceiverId*

Methods

The *Channel* object does not contain any methods.

Properties

The *Channel* object contains the following properties:

- *AltPathType*
- *AppDevices*
- *BitmapFilePath*
- *ClassId*
- *Delay*
- *Description*
- *Extensions*
- *Handle*
- *IconFilePath*
- *MaxPriority*
- *Name*
- *Parent*
- *RouterDevices*
- *TransceiverId*

AltPathType

Summary	Specifies whether the channel requires an alternate path so that it can broadcast on separate frequencies. In some cases, the use of alternate frequencies improves communication on a channel. Some transceivers broadcast on two frequencies for greater signal reliability (for example, the PLT-22). This property allows the user to explicitly inform LNS that a channel has (or does not have) an alternate frequency.
Availability	Local, full, and lightweight clients.
Syntax	*altPathType* = *channelObject*.**AltPathType** **Element** — **Description** *channObject* — The *Channel* object to be acted on. *altPathType* — The alternate path type. The enumerated values for this property, which are contained in the *ConstChannelAltPathType* constant, are as follows: **0 lcaChannelDefaulAltPathType** Channels with their *TransceiverId* property set to **lcaXcvrPL_20A (15)**, **lcaXcvrPL_20A_AltClockRate (14), lcaXcvrPL_20C (16)**, or **lcaXcvrPL_20N (17)**, will require the alternate path, other channels will not. The *TransceiverType* property does not distinguish between PLT-20, PLT-21 and PLT-22. This is the default and the recommended setting.

	1 lcaChannelAltPathNotRequired This channel does not require use of the alternate path. This value should be set when a channel uses only PL-20s and PL-21s. These power line transceivers do not use the alternate path, but look the same to LNS as the PL-22, which does use the alternate path. **2 lcaChannelAltPathRequired** This channel requires use of both the primary and alternate paths. This value should be set when using a custom transceiver that uses alternate frequencies.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

AppDevices

Summary	Contains the *AppDevices* collection object associated with the specified *Channel* object. This property represents all the devices on the channel.
Availability	Local, full, and lightweight clients.
Syntax	*appDevicesCollection* = *object*.**AppDevices** **Element** — **Description** *appDevicesCollection* — The AppDevices collection returned. *object* — The *Channel* object to be acted on.
Data Type	*AppDevices* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.

Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** **Description** *bmpFilePath* The bitmap path and file name. *object* The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Channel* object in the *ConstClassIds* constant: **12 lcaClassIdChannel** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Delay

Summary	Represents the average number of milliseconds required for a packet to get onto the channel once queued. This property allows OpenLNS applications to specify the number of milleseconds expected to send a message and receive an acknowledgment on the specified channel, so that automatic timer calculations made by OpenLNS can be affected accordingly. When this property contains the default value of 0, the delay used will be equivalent to the time required for two packet cycles, based on the average packet

	size and channel transceiver type. When you write to this property, you should note that the delays for transactions on a given channel must be calculated as round-trip delays. Make sure you set this property to a value that is based on the amount of time it will take for a request message to be sent on the channel, and for the response message to be sent back on the channel. You should also note that you can set the expected delay for a message to be sent to a specific device by writing to the *Delay* property of the *AppDevice* object.
Availability	Local, full, and lightweight clients.
Syntax	*delayValue* = *channelObject*.**Delay** **Element** — **Description** *delayValue* — The delay associated with the channel, in milliseconds. The valid range of this property is 0 to 65,535. *channelObject* — *Channel* object to be acted upon.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *Channel* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *AppDevice* object. *object* — The *AppDevice* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *Channel*. This property returns an Extensions collection. The objects in this collection represent user data reserved for

	manufacturers. Each object is identified with a unique identifier set by the manufacturer
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.**Extensions** **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose Extensions collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *Channel* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property of the *Channels* collection object. Some OpenLNS Objects are tracked internally by the OpenLNS Server using a unique handle. Handles may be used with the *ItemByHandle* method of the *Channels* collection object as an alternative means of fetching *Channel* objects.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the object. *object* — The object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (*.**ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** — **Description** *IconFilePathFileName* — Icon file and path name

	object The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

MaxPriority

Summary	Specifies the total number of priority slots allowed on a channel.
Availability	Local, full, and lightweight clients.
Syntax	*numPrioritySlots* = *channelObject*.**MaxPriority** **Element** — **Description** *numPrioritySlots* — The total number of priority slots. This value may be between 0 to **lcaNodePriorityMax (127)**. Generally, this property only needs to for channels based upon custom transceivers. *channelObject* — The *Channel* object to be acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters. The default name for a discovered channel takes the following form: *Channel_<objectId>*

Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

RouterDevices

Summary	Returns the *Routers* collection for the specified *Channel*. This collection contains all the *Router* objects attached to the specified *Channel* object.
Availability	Local, full, and lightweight clients.
Syntax	*routersCollection* = o*bject*.**RouterDevices** **Element** — **Description** *object* — The *Channel* object to be acted on.

	routersCollection The *Routers* collection returned.
Data Type	*Routers* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TransceiverId

Summary	Contains the current transceiver ID of the channel.
Availability	Local, full, and lightweight clients.
Syntax	*idValue* = *channelObject*.**TransceiverId** **Element** **Description** *idValue* The transceiver ID value. The TransceiverId property can contain a standard transceiver ID (1-29) or a custom ID (30). The enumerated values for this property, which correspond to standard transceiver types, are stored in the *ConstTransceiverId* constant. The values are as follows: **1 lcaXcvrTP_XF_78** TP/XF-78 LONWORKS transceiver. **3 lcaXcvrTP_XF_1250** TP/XF-1250 LONWORKS transceiver. **4 lcaXcvrTP_FT_10** The TP/FT-10 channel used with FTT-10, FTT-10A, LPT-10, LPT-11 and Free Topology Smart Transceivers. **5 lcaXcvrTP_RS485_39** The TP/RS485-39 LONWORKS transceiver. **7 lcaXcvrRF_10** RF-10 LONWORKS transceiver. **8 lcaXcvrPL_30** PLT-30 LONWORKS transceiver. **9 lcaXcvrPL_10** PL-10 LONWORKS transceiver. **10 lcaXcvrTP_RS485_625** TP/RS485-625 LONWORKS

		transceiver. **11 lcaXcvrTP_RS485_1250** TP/RS485-1250 LonWork transceiver. **12 lcaXcvrTP_RS485_78** TP/RS-485-78 LONWORKS transceiver. **14 lcaXcvrPL_20A_LOW** The low-powered PL-20A-LOW channel used with PLT-20, PLT-21, PLT-22, PLT-22A and Power Line Smart Transceivers using the CENELEC A-band. **15 lcaXcvrPL_20A** The PL-20A channel used with PLT-20, PLT-21, PLT-22, PLT-22A and Power Line Smart Transceivers using the CENELEC A-band. **16 lcaXcvrPL_20C** The PL-20C channel used with PLT-20, PLT-21, PLT-22, PLT-22A and Power Line Smart Transceivers using the CENELEC C-band. **17 lcaXcvrPL_20N** The PL-20N channel used with PLT-20, PLT-21, PLT-22, PLT-22A, and Power Line Smart Transceivers configured to not use the CENELEC C-band protocol. **24 lcaXcvrFO_10** FO-10 LONWORKS transceiver. **25 lcaXcvrIP_10L** IP-10 LAN Low Latency LONWORKS transceiver. **26 lcaXcvrIP_10W** IP-10 WAN High Latency LONWORKS transceiver. **27 lcaXcvrDC_78** DC-78 LONWORKS transceiver. **28 lcaXcvrDC_625** DC-625 LONWORKS transceiver.

	29 lcaXcvrDC_1250 DC-1250 LONWORKS transceiver. *channelObject* *Channel* object to be acted upon.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Channels

A Channels object is a collection of *Channel* objects. The instance of this collection accessed through the Network object contains all of the Channel objects in the network.

Description	Represents a collection of *Channel* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Network* object
Default Property	*Item*.
Methods	• *Add* • *ItemByHandle* • *Remove*
Properties	• *ClassId* • *Count* • *FilterType* • *Item* • *Parent* • *RefChannel1* • *RefChannel2* • *RefTransceiverType* • *_NewEnum*

Methods

The C*hannels* object contains the following methods.

- *Add*
- *ItemByHandle*
- *RemoveRemove*

Add

Summary	Defines a new *Channel* object. Channels only need to be defined when router devices are added or for when automatic channel determination is not used when adding devices (see the *Add* method of the *AppDevices* object for more information).
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Add** *name*

	Element	Description
	channelObject	The newly defined channel object.
	channelsObject	The Channels collection object.
	channelName	A String containing the name of the channel.
	transceiverId	A Long value indicating the transceiver type for the channel. The enumerated values for this element, which correspond to standard transceiver types, are stored in the *ConstTransceiverId* constant. The values are as follows: **1 lcaXcvrTP_XF_78** TP/XF-78 LONWORKS transceiver. **3 lcaXcvrTP_XF_1250** TP/XF-1250 LONWORKS transceiver. **4 lcaXcvrTP_FT_10** TP/FTT-10 and FTT-11 LONWORKS transceiver. **5 lcaXcvrTP_RS485_39** TP/RS485-29 LONWORKS transceiver. **7 lcaXcvrRF_10** RF-10 LONWORKS transceiver. **9 lcaXcvrPL_10** PL-10 LONWORKS transceiver. **10 lcaXcvrTP_RS485_625** TP/RS485-625 LONWORKS transceiver. **11 lcaXcvrTP_RS485_1250** TP/RS485-1250 LonWork transceiver. **12 lcaXcvrTP_RS485_78** TP/RS-485-78 LONWORKS transceiver. **14 lcaXcvrPL_20A_LOW** A low-powered version of PLT-20, PLT-21, and PLT-22 LONWORKS transceivers using CENELEC A-band. **15 lcaXcvrPL_20A** PLT-20, PLT-21, and PLT-22 LONWORKS transceivers using

	CENELEC A-band. **16 lcaXcvrPL_20C** PLT-20, PLT-21, and PLT-22 LONWORKS transceivers using CENELEC C-band. **17 lcaXcvrPL_20N** PLT-20, PLT-21, and PLT-22 LONWORKS transceivers using Non-CENELEC band. **18 lcaXcvrPL_30** PLT-30 LONWORKS transceiver. **24 lcaXcvrFO_10** FO-10 LONWORKS transceiver. **25 lcaXcvrIP_10L** IP-10 LAN Low Latency LONWORKS transceiver. **26 lcaXcvrIP_10W** IP-10 WAN High Latency LONWORKS transceiver. **27 lcaXcvrDC_78** DC-78 LONWORKS transceiver. **28 lcaXcvrDC_625** DC-625 LONWORKS transceiver. **29 lcaXcvrDC_1250** DC-1250 LONWORKS transceiver.
Added to API	Prior to LNS Release 3.0.

ItemByHandle

Summary	Retrieves a *Channel* object, specified by its handle, from an *Channels* collection. The *Channel* object to be retrieved must be specified by its handle.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.**ItemByHandle**(*handle*) **Element** — **Description** *object* — The object retrieved from the collection. *coll* — The collection object. *handle* — A Long value specifying the handle of the object to retrieve.
Added to API	LNS Release 3.0.

Remove

Summary	Removes a channel from the network. A channel may only be removed if there are no devices or routers assigned to it.
Availability	Local, full, and lightweight clients.
Syntax	*channelsColl*.**Remove** *indexName* **Element** — **Description** *channelsColl* — The *Channels* collection object containing the channel to be removed. *indexName* — A Long value specifying the collection index of the *Channel* object to remove, or a String value specifying the name of the *Channel* object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *Channels* object contains the following properties:

- *ClassId*
- *Count*
- *FilterType*
- *Item*
- *Parent*
- *RefChannel1*
- *RefChannel2*
- *RefTransceiverType*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Channels* object in the *ConstClassIds* constant: **13 lcaClassIdAppChannels** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property is read only for *Error* objects.

FilterType

Summary	Enables a subset of the items in the *Channels* collection to be selected.
	The filter property can be set to allow queries on selected groups of objects in the *Channels* collection. Set the *FilterType* property to the appropriate constant values before accessing any objects in the collection.
	You can use *RefChannel1* and *RefChannel2* properties included in the *Channels* object to help search a *Channels* collection. These variants have meaning in the context of particular filters.
	To use channel filtering, a populated channel object must first be obtained. For example:
	`Set AllChannels = Network.Channels`
	`Set FilteredChannels = AllChannels`
	`FilteredChannels.FilterType = lcaChannelFilterAdjacent`
	`'Print list of channels adjacent to channel named "Room1".`
	`'Assume PrintChannels 'function has been defined.`
	`Set FilteredChannels.RefChannel1 = AllChannels.Item("Room1")`
	`PrintChannels FilteredChannels`

Availability	Local, full, and lightweight clients.

Syntax	*filterValue* = *channelsCollection*.**FilterType** **Element** — **Description** *filterValue* — The active filter type. The valid values for this property, which are contained in the *ConstChannelFilterTypes* constant, are as follows: **0 lcaChannelFilterNone** All objects in the collection are shown. **1 lcaChannelFilterAdjacent** Only channels adjacent to *RefChannel1* (as Channel object) are shown. **2 lcaChannelFilterPath** Only channels connecting *RefChannel1* (as Channel object) and *RefChannel2* (as Channel object) are shown. **3 lcaChannelFilterTransceiver** Only channels which have the transceiver ID specified in *RefTransceiverType* (as integer) are shown. *channelsCollection* — The Channels collection object.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Item

Summary	Returns an object from a collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve an object in collections that contain objects with the *Name* property by passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *Channel* object retrieved from the collection.

	collObject	The *Channels* collection object to be acted on.
	index	A Long type specifying the ordinal index of the *Channel* object to be retrieved.
	stringExpression	A string type specifying the name of the *Channel* object to be retrieved.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Parent

Summary	Returns the object that spawned the current child object.	
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.	
Syntax	*parentObject* = *object*.**Parent**	
	Element	**Description**
	parentObject	The object that is the parent of the specified *object*.
	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.	

RefChannel1

Summary	Contains filter data enabling access to selected channels within the channels collection. The meaning of this property depends on the *FilterType* property of the Channels collection object: • If the *FilterType* property is set to **lcaChannelFilterAdjacent**, the Channels collection object will access all channels adjacent—directly connected by a router—to the channel specified by *RefChannel1*.

	• If the *FilterType* property is set to **lcaChannelFilterPath**, the *Channels* collection object will access all channels connecting the channels specified by *RefChannel1* and *RefChannel2*. *RefChannel1* is not used for any other value of the *FilterType* property.
Availability	Local, full, and lightweight clients.
Syntax	*filterValue* = *channelsColl*.**RefChannel1** **Element** — **Description** *channelObject* — The *Channel* object to be used as a reference. *channelsColl* — The *Channels* collection object.
Data Type	*Channel* object.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

RefChannel2

Summary	Contains filter data enabling access to selected channels within the channels collection. The meaning of this property depends on the *FilterType* property of the Channels collection object. If the *FilterType* property is set to **lcaChannelFilterPath**, the *Channels* collection object will access all channels connecting the channels specified by *RefChannel1* and *RefChannel2*. *RefChannel2* is not used for any other value of the *FilterType* property.
Availability	Local, full, and lightweight clients.
Syntax	*filterValue* = *channelsColl*.**RefChannel2** **Element** — **Description** *channelObject* — The *Channel* object to be used as a reference. *channelsColl* — The *Channels* collection object.
Data Type	Channel object.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

RefTransceiverType

Summary	Contains filter data enabling access to selected *Channel* objects within the *Channels* collection. This property is expected to be correctly set and is used only when the *Channels* collection object's *FilterType* property is

	set to **lcaChannelFilterTransceiver**. When this is the case, the *Channels* collection object will access the *Channel* objects with the *TransceiverId* property equal to this property.
Availability	Local, full, and lightweight clients.
Syntax	*rtTypeValue* = *channelsColl*.**RefTransceiverType** **Element** **Description** *channelsColl* The *Channels* collection to be acted on. *rtTypeValue* Transceiver ID value
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** **Description** *retrievedObject* An expression that evaluates to the object retrieved. *collObject* An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

ComponentApp

This object describes a LONWORKS plug-in command. *ComponentApps* collection objects are contained in the *ObjectServer*, *System*, and *DeviceTemplate* objects. An LCA director application can determine what plug-in commands are available by examining the *ComponentApp* objects and can launch the plug-in using the *RegisteredServer* property. Plug-in commands to be used by all networks are listed in the *ComponentApps* collection in the *ObjectServer* object. Plug-in commands that are specific to a network are listed in the *System* objects. Plug-in commands that are specific to a device type are listed in the *DeviceTemplate* objects.

The following table summarizes the *ComponentApp* object.

Description	A LONWORKS plug-in command.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*ComponentApps* collection object.
Default Property	*Name*.
Methods	None.
Properties	• *ClassId* • *CommandId* • *ComponentClassId* • *DefaultAppFlag* • *Description* • *ManufacturerId* • *Name* • *Parent* • *RegisteredServer* • *VersionNumber*

Methods

The *ComponentApps* object does not contain any methods.

Properties

The *ComponentApp* object contains the following properties:

- *ClassId*
- *CommandId*
- *ComponentClassId*
- *DefaultAppFlag*
- *Description*
- *ManufacturerId*
- *Name*
- *Parent*
- *RegisteredServer*
- *VersionNumber*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ComponentApp* object in the *ConstClassIds* constant: **30 lcaClassIdComponentApp** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

CommandID

Summary	Contains a command to be sent to the plug-in by the director application.
Availability	Local, full, lightweight, and independent clients.
Syntax	*commIdValue* = *caObject*.**CommandId** **Element** — **Description** *caObject* — The *ComponentApp* object to be acted on. *commIdValue* — The command ID for the specified command as a Long. This may be a standard command or a user-defined command. Standard commands are included in the *ConstCommandId* constant. User-defined commands are application specific. Each plug-in registers its commands in the *ComponentApps* collections. The values for standard commands are as follows:

	1 lcaCommandIdNew **2 lcaCommandIdEditSource** **10 lcaCommandIdBuildImage** **11 lcaCommandIdCommission** **12 lcaCommandIdLoad** **13 lcaCommandIdConfigure** **14 lcaCommandIdCalibrate** **15 lcaCommandIdConnect** **20 lcaCommandIdBrowse** **21 lcaCommandIdMonitor** **22 lcaCommandIdControl** **23 lcaCommandIdReport** **24 lcaCommandIdProperties** **30 lcaCommandIdOnline** **31 lcaCommandIdOffline** **32 lcaCommandIdReset** **33 lcaCommandIdTest** **34 lcaCommandIdWink** **35 lcaCommandIdDebug** **40 lcaCommandIdUninstall** **41 lcaCommandIdReplace** **50 lcaCommandIdRegister** **51 lcaCommandIdUnregister** **60 lcaCommandIdRecover** **61 lcaCommandIdMonitorRecovery** **10000 lcaCommandUserStart**
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ComponentClassID

Summary	Specifies an object type. This is used by director applications to determine the class of object specified in the object name passed to a plug-in.
Availability	Local, full, lightweight, and independent clients.
Syntax	*classIdValue* = *object*.**ComponentClassId** **Element** **Description** *classIdValue* The enumerated value of the class ID. The values for this property, which are included in the *ConstClassIds* constant, are as follows: **0 lcaClassIdObjectServer** **1 lcaClassIdNetwork** **2 lcaClassIdNetworks** **3 lcaClassIdSystem** **4 lcaClassIdSystems** **5 lcaClassIdSubsystem**

	6 lcaClassIdSubsystems **7 lcaClassIdAppDevice** **8 lcaClassIdAppDevices** **9 lcaClassIdRouter** **10 lcaClassIdRouters** **11 lcaClassIdRouterSide** **12 lcaClassIdChannel** **13 lcaClassIdChannels** **14 lcaClassIdNetworkInterface** **15 lcaClassIdNetworkInterfaces** **16 lcaClassIdSubnet** **17 lcaClassIdSubnets** **18 lcaClassIdConnections** **19 lcaClassIdInterface** **20 lcaClassIdInterfaces** **21 lcaClassIdTemplateLibrary** **22 lcaClassIdNetworkVariable** **23 lcaClassIdNetworkVariables** **24 lcaClassIdMessageTag** **25 lcaClassIdMessageTags** **26 lcaClassIdConfigProp** **27 lcaClassIdConfigProps** **28 lcaClassIdLonMarkObject** **29 lcaClassIdLonMarkObjects** **30 lcaClassIdComponentApp** **31 lcaClassIdComponentApps** **32 lcaClassIdHardwareTemplate** **33 lcaClassIdHardwareTemplates** **34 lcaClassIdBuildTemplate** **35 lcaClassIdBuildTemplates** **36 lcaClassIdDeviceTemplate** **37 lcaClassIdDeviceTemplates** **38 lcaClassIdProgramTemplate** **39 lcaClassIdProgramTemplates** **40 lcaClassIdNetworkServiceDevice** **41 lcaClassIdNetworkServiceDevices** **42 lcaClassIdConnectDescTemplate** **43 lcaClassIdConnectDescTemplates** **44 lcaClassIdError** **45 lcaClassIdLonMarkAlarm** **46 lcaClassIdObjectStatus** **47 lcaClassIdNetworkVariableField** **48 lcaClassIdDetailInfo** **49 lcaClassIdDataValue** **50 lcaClassIdExtension** **51 lcaClassIdExtensions** **52 lcaClassIdRecoveryStatus** **53 lcaClassIdCreditInfo** **54 lcaClassIdAccount** **55 lcaClassIdAccounts** **56 lcaClassIdBufferConfiguration** **57 lcaClassIdFileTransfer** **58 lcaClassIdAlias**

	59 lcaClassIdAliases **69 lcaClassIdPingIntervals** **70 lcaClassIdApplication** **71 lcaClassIdTestInfo** **72 lcaClassIdDataPoint** **73 lcaClassIdFormatSpec** **74 lcaClassIdMonitorSet** **75 lcaClassIdMonitorSets** **76 lcaClassIdMsgMonitorOptions** **77 lcaClassIdMsgMonitorPoint** **78 lcaClassIdMsgMonitorPoints** **79 lcaClassIdNvMonitorOptions** **80 lcaClassIdNvMonitorPoint** **81 lcaClassIdNvMonitorPoints** **82 lcaClassIdSourceAddress** **83 lcaClassIdLdrfLanguage** **84 lcaClassIdLdrfLanguages** **85 lcaClassIdServiceStatus** **86 lcaClassIdUpgradeStatus** **87 lcaClassIdUpgradeInfo** **88 lcaClassIdUpgradeInfos** **89 lcaClassIdDatabaseValidationReport** **90 lcaClassIdDatabaseValidationError*Summary*** **91 lcaClassIdDatabaseValidationErrorSummaries** **92 lcaClassIdDatabaseValidationErrorInstance** **93 lcaClassIdNetworkResources** **94 lcaClassIdTypeSpec** **95 lcaClassIdFormatLocale** **96 lcaClassIdFormatLocales** **97 lcaClassIdActivationLicense** **-1 lcaClassIdUnknown** *object* The object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

DefaultAppFlag

Summary	Specifies that a plug-in command is the default command for the object type. Director applications can use this flag to launch component applications as a default action such as double-clicking on an icon representing the object or selecting a command for the icon.
Availability	Local, full, lightweight, and independent clients.
Syntax	*defaultAppFlag* = *compAppObject*.**DefaultAppFlag** **Element** **Description** *defaultAppFlag* The flag value. A TRUE value indicates that this plug-in command

	implements the default action *compAppObject* — The *ComponentApp* object to be acted on.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *ComponentApps* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *ComponentApps* object. *object* — The *ComponentApps* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

ManufacturerID

Summary	Reads the manufacturer ID assigned to a device template or component application. This property is the manufacturer ID field of the standard program ID. See the *LonMark Application Layer Interoperability Guidelines* for more information.
Availability	Local, full, lightweight, and independent clients.
Syntax	*idValue* = *object*.**ManufacturerId** **Element** — **Description** *idValue* — The component application or device template's manufacturer ID. *object* — The *ComponentApp* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of a *ComponentApp* object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the *ComponentApp* object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

RegisteredServer

Summary	Contains the plug-in name that is stored in the Windows registry for this component application. Component applications must follow the Windows component registration guidelines.
Availability	Local, full, lightweight, and independent clients.
Syntax	*serverName* = *compAppObject*.**RegisteredServer** **Element** — **Description** *compAppObject* — The *ComponentApp* object to be acted on. *serverName* — The registered name of the plug-in as a string.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

VersionNumber

Summary	The version number of the *ComponentApp* object.
Availability	Local, full, lightweight, and independent clients.
Syntax	*versionValue* = *compAppObject*.**VersionNumber** **Element** — **Description** *versionValue* — The component application version number. *compAppObject* — The *ComponentApp* object from which to get the information.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ComponentApps

A *ComponentApps* object is a collection of *ComponentApp* objects. *ComponentApps* collection objects are contained in the *ObjectServer*, *System*, *DeviceTemplate*, and *LonMarkObject* objects. An LCA director application can determine what plug-in commands are available by examining the *ComponentApp* objects and can launch the plug-in using the *RegisteredServer* property:

- Plug-in commands to be used by all networks are listed in the *ComponentApps* collection in the *ObjectServer* object.
- Plug-in commands that are specific to a network are listed in the *System* objects.
- Plug-in commands that are specific to a device type are listed in the *DeviceTemplate* objects.

Note that the *Interface* property of all *AppDevice* objects contain a *ComponentApps* collection; however, the behavior of this collection is unspecified.

The following table summarizes the *ComponentApps* object.

Description	A collection of objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*DeviceTemplate* object *LonMarkObject* *ObjectServer* object *System* object
Default Property	*Item*
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *ComponentApps* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	Defines a new *ComponentApp* object.
Availability	Local, full, lightweight, and independent clients.
Syntax	*compAppObject* = *compAppsColl*.**Add** (*compAppName, registeredServer, classId*)

Element	Description
compAppObject	The newly defined *ComponentApp* object.
compAppsColl	The *ComponentApps* collection object.
compAppName	A String containing the name of the plug-in command. The name should be a descriptive name suitable for display as a menu selection.
registeredServer	A String containing the registered (OLE) automation server name for the plug-in containing the command.
classId	An Integer value indicating the type of object with which the *ComponentApp* object is to be associated. See the *ComponentClassID* property in

	the *ComponentApp* object for the possible values.
Added to API	Prior to LNS Release 3.0.

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed. *name* — A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *ComponentApps* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ComponentApps* object in the *ConstClassIds* constant: **31 lcaClassIdComponentApps** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.

Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an object from a collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve an object in collections that contain objects with the *Name* property by passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The object retrieved from the collection. *collObject* — The collection object to be acted on. *index* — A Long type specifying the ordinal index of the object to retrieve. *stringExpression* — A string type specifying the name of the object to retrieve.

Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* **=** *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.

Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** **Description** *retrievedObject* An expression that evaluates to the object retrieved. *collObject* An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

ConfigProperties

A *ConfigProperties* object is a collection of *ConfigProperty* objects. Configuration properties may be associated with a *device*, *LonMark object*, or *network variable*. The following table summarizes the *ConfigProperties* object.

Description	A collection of *ConfigProperty* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Interface* object *LonMarkObject* object *NetworkVariable* object
Default Property	*Item*
Methods	• *ItemByHandle*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *ConfigProperties* object contains the following method: *ItemByHandle*.

ItemByHandle

Summary	Retrieves a *ConfigProperty* object, specified by its handle, from a *ConfigProperties* collection. The *ConfigProperty* object to be retrieved must be specified by its handle.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.**ItemByHandle**(*handle*) **Element** **Description** *object* The *ConfigProperty* object retrieved from the collection. *coll* The collection object.

	handle	A Long value specifying the handle of the *ConfigProperty* object to be retrieved.
Added to API	LNS Release 3.0.	
Added to API	Prior to LNS Release 3.0.	

Properties

The *ConfigProperties* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ConfigProperties* object in the *ConstClassIds* constant: **27 lcaClassIdConfigProps** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.

Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *ConfigProperty* object from a *ConfigProperties* collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve an object in collections that contain objects with the *Name* property by passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** **Description** *retrievedObject* The *ConfigProperty* object retrieved from the *ConfigProperties* collection. *collObject* The *ConfigProperties* collection object to be acted on. *index* A Long type specifying the ordinal index of the *ConfigProperty* object to be retrieved. *stringExpression* A string type specifying the name of the *ConfigProperty* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.

Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

ConfigProperty

A *ConfigProperty* object represents a single configuration property, which may be either a standard configuration property type (SCPT) or a user configuration property type (UCPT). The configuration property can be implemented on the target *AppDevice* as a configuration network variable or as part of a configuration file (using sequential access, sequential/random access, or direct memory access). Configuration properties may be associated with a *device*, *LonMark object*, or *network variable*.

Description	A single SCPT or UCPT.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*ConfigProperties* collection object
Default Property	*Name*
Methods	• *GetDataPoint* • *GetRawValuesEx* • *SetRawValuesEx* • *SetValueToUnknown*
Properties	• *AppliesTo* • *AppliesToType* • *ByteLength* • *ClassId* • *ConfigFileOffset* • *ConfigFileType* • *ConfigNv* • *ConfigNvIndex* • *ConstantAttribute* • *Description* • *DeviceSpecificAttribute* • *Dimension* • *FlagsByte* • *FormatName* • *FuncProfileDescription* • *FuncProfileName* • *FuncProfileProgrammaticName* • *Handle* • *ImplementationType* • *Mode* • *Name* • *Parent* • *Precision* • *TypeInherits* • *TypeIndex* • *ValueStatus*

Methods

The *ConfigProperties* object contains the following methods:

- GetDataPoint
- *GetRawValuesEx*
- *SetRawValuesEx*
- *SetValueToUnknown*

GetDataPoint

Summary	Returns a *DataPoint* object that can be used to read or write the configuration property value. Use this method to read and write configuration property values instead of the deprecated *ValueFromDevice*, *RawValueFromDevice*, *Value* and *RawValue* properties. If the data point created applies to an element in an array, the element's index will be stored in the data point's *SourceIndex* property. You can write to this property to apply the data point to another element of the array later (see the *SourceIndex* property for more information). To read or write to an entire configuration property array at once, you can use the *GetRawValuesEx* and *SetRawValuesEx* methods.
Availability	Local, full, and lightweight clients.
Syntax	*dpObject=configProp*.**DataPoint** *index, options* **Element** — **Description** *dpObject* — The *DataPoint* object returned. *configProp* — The *ConfigProperty* object being acted upon. *index* — Specify which element in the configuration property array to which the data point applies. Configuration properties may be stored in array. In this case, you need to create a separate data point to read and write to each element in the array. If the *ConfigProperty* is not a configuration property array, specify 0 as the *index* element. *options* — Specify how LNS reconciles differences between the value of the *ConfigProperty* in the OpenLNS database and the physical application device when you read or write to the *DataPoint*. The possible values for this element, which are stored in the *ConstDataSourceOptions* constant, are as follows: **0 lcaDataSourceOptionsNormal**

		The value of the *ConfigProperty* will be updated in the OpenLNS database and in the physical device each time you write to the value of the *DataPoint*. If the source *ConfigProperty* is device-specific, the value will be read directly from the device when you read the value of the *DataPoint*. If the source *ConfigProperty* is not device-specific, the value will be read from the database, as long as it is stored there. If its value does not exist in the database, then the value will read directly from the device, as long as the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0)**. If the source *ConfigProperty* is not device-specific, the value is not in the database, and the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**, then an exception will be thrown when you read the value of the *DataPoint*. **1 lcaDataSourceOptionsFromDevice** The data point value is always matched to the value of the source *ConfigProperty* in the physical device. You can use this information to synchronize the value of a *ConfigProperty* in the OpenLNS database with the value stored in the physical device. To do so, read the value of a data point created with this option set. Then, set the SourceOptions property of the data point to **lcaDataSourceOptionsDatabaseOnly (2)**, and call the *Write* method. The value of the source *ConfigProperty* in the OpenLNS database will then match the value of the configuration property on the physical device. **2 lcaDataSourceOptionsDatabaseOnly** The data point value is always read from the OpenLNS database. When you write to the data point, the new value will be written to the *ConfigProperty* in the OpenLNS database only, and not to the physical device. Writing to a *DataPoint* with the SourceOptions property set to this

		value is recommended only when updating the database with a value that has just been read from the device. For more information on this, see the description of the **lcaDataSourceOptionsFromDevice (1)** value. If you read the value of the data point, and the value does not exist in the OpenLNS database, then the **NS, #113 lcaErrNsCpValueNotFound** exception is thrown. **3 lcaDataSourceOptionsTypeDefaultValue** The data point value is set to the default value of configuration properties using the same type as the source configuration property. The default value is generally read from the functional profile template on the device containing the configuration property, or from the type definition for this configuration property type. Data points created with this option set are read-only. Note that this value represents the "type default", as defined in the resource files. The default value of a given configuration property may differ from the default value of its type, since the default configuration property values for a given template are defined in the external interface file and can be set from the current values in the device.
Added to API	LNS Release 3.20.	

GetRawValuesEx

Summary	Retrieves the raw values of a range of configuration property elements as a raw byte array. The configuration property values are returned as a Variant containing a one-dimensional byte array (MFC type VT_UI1 | VT_ARRAY). The application is responsible for parsing the byte array and "reformatting" the raw values.	
Availability	Local, full, and lightweight clients.	
Syntax	*rawValues = cpObject.GetRawValuesEx index, count, options*	
	Element	**Description**
	rawValues	A *Variant* containing the returned raw values.
	cpObject	The *ConfigProperty* object being acted upon.
	index	Specify the first element in the array to be

		retrieved. If you specify an invalid index, an **LCA, #90 lcaValueOutOfRange** exception is thrown.
	count	Specify the number of elements to be retrieved. If this number exceeds the value of the *ConfigProperty* object's *Dimension* property, an **LCA, #90 lcaValueOutOfRange** exception is thrown.
	options	Use this element to determine whether OpenLNS will read the raw values of the *ConfigProperty* array elements from the OpenLNS database or the physical application device. The possible values for this element, which are stored in the *ConstDataSourceOptions* constant, are as follows: **0 lcaDataSourceOptionsNormal** The raw value array will be read directly from the device if the *ConfigProperty* is device-specific. If the *ConfigProperty* is not device-specific, the raw value array will be read from the database, as long as it is stored there. If the value of the *ConfigProperty* does not exist in the database, then the raw value value array will read directly from the device, as long as the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0)**. If the *ConfigProperty* is not device-specific, its value is not in the database, and the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**, then an exception will be thrown when you invoke the *GetRawValuesEx* method. **1 lcaDataSourceOptionsFromDevice** The raw value array is read directly from the device. **2 lcaDataSourceOptionsDatabaseOnly** The raw value array is read from the OpenLNS database.

Added to API	LNS Release 3.20.

SetRawValuesEx

Summary	Sets the raw value for a range of elements in an array configuration property. You can specify configuration property values as either a byte array or a Variant containing a byte array (MFC type VT_UI1 | VT_ARRAY). The application is responsible for creating the raw byte array that corresponds to the desired configuration property values.
Availability	Local, full, and lightweight clients.
Syntax	*cpObject.SetRawValuesEx index, count, array, options* **Element** — **Description** *cpObject* — The *ConfigProperty* object to be acted on. *index* — Specify the first element in the array to write to. If you specify an invalid index, an **LCA, #90 lcaValueOutOfRange** exception is thrown. *count* — Specify the number of elements to write to. If this number exceeds the value of the *ConfigProperty* object's *Dimension* property, an **LCA, #90 lcaValueOutOfRange** exception is thrown. *array* — A *Variant* containing the raw values to be written to the *ConfigProperty* object. *options* — Use this element to determine whether OpenLNS will write new raw values of the *ConfigProperty* array elements to the OpenLNS database or to the physical application device. The possible values for this element, which are stored in the *ConstDataSourceOptions* constant, are as follows: **0 lcaDataSourceOptionsNormal** Writes the raw value array to the physical device. **2 lcaDataSourceOptionsDatabaseOnly** Writes the raw value array to the OpenLNS database. Calling the *SetRawValuesEx* method with this option set is recommended only to update the database with a value that has

	just been read from the device (for example, create a data point with the **lcaDataSourceOptionsFromDevice (1)** option set, then set the SourceOptions property of the data point to this value and call the *SetRawValuesEx* method).
Added to API	LNS Release 3.20.

SetValueToUnknown

Summary	Sets the value of the *ConfigProperty* object to an unknown state. You can use this method to clear the value of a *ConfigProperty* object from the OpenLNS database. When you call this method, the value of the *ConfigProperty* will be cleared from the OpenLNS database. In addition, all pending updates to the *ConfigProperty* object will be cleared. You should note that this affects how OpenLNS will manage the configuration property in read and download operations. In addition, you can use the *UploadConfigProperties* method to read all unknown configuration property values into the OpenLNS database, and you can use the *DownloadConfigProperties* method to set all unknown configuration property values to their defaults. You can read the current state of the value of a *ConfigProperty* object in the OpenLNS database with the *ValueStatus* property. If this property is set to **lcaConfigPropertyValueMgmtStatusUnknown (0)**, then the value of the *ConfigProperty* object is already set to an unknown state. In this case, use of the *SetValueToUnknown* method will have no effect on the *ConfigProperty*.
Availability	Local, full, and lightweight clients.
Syntax	*cpObject*.**SetValueToUnknown** **Element** — **Description** *cpObject* — The *ConfigProperty* object to be acted on.
Added to API	LNS Release 3.20.

Properties

The *ComponentApps* object contains the following properties:

- *AppliesTo*
- *AppliesToType*
- *ByteLength*
- *ClassId*
- *ConfigFileOffset*
- *ConfigFileType*
- *ConfigNv*
- *ConfigNvIndex*

- *ConstantAttribute*
- *Description*
- *DeviceSpecificAttribute*
- *Dimension*
- *FlagsByte*
- *FormatName*
- *FuncProfileDescription*
- *FuncProfileName*
- *FuncProfileProgrammaticName*
- *Handle*
- *ImplementationType*
- *Mode*
- *Name*
- *Parent*
- *Precision*
- *TypeInherits*
- *TypeIndex*
- *ValueStatus*

AppliesTo

Summary	Returns the collection of objects to which the configuration property applies. Each configuration property applies to the interface of a *AppDevice* or *DeviceTemplate* as a whole, a collection of *LonMarkObjects*, or a collection of *NetworkVariable* objects. This property returns a (generic) collection of objects to which the configuration property applies. To determine the type of the objects to which the configuration property applies, use the *AppliesToType* property.
Availability	Local, full, and lightweight clients.
Syntax	*coll* = *cpObject*.**AppliesTo** **Element** — **Description** *nvObject* — The collection of objects to which this configuration property applies. The objects will be of the type specified by the *AppliesToType* property. *cpObject* — The *ConfigProperty* object to be acted on.
Data Type	Collection.
Read/Write	Read only.
Added to API	OpenLNS.

AppliesToType

Summary	Returns a constant identifying the type of objects to which the *ConfigProperty* object applies. Each configuration property applies to the interface of a

<table>
<tr><td></td><td>AppDevice or DeviceTemplate as a whole, a collection of LonMarkObjects, or a collection of NetworkVariable objects. This property returns a (generic) collection of objects to which the configuration property applies.

This property indicates ther type of objects the configuration property applies. To access the actual objects to which the configuration property applies, use the AppliesTo property.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>objectType = cpObject.AppliesToType

Element — Description

objectType — The type of objects to which this configuration property applies. The possible object types, which are contained in ConstCpAppliesToTypes constant, are as follows:

1 lcaCpAppliesToInterface

The configuration property applies to an AppDevice or DeviceTemplate as a whole. The object returned by the AppliesTo property will be a collection of Interface objects belonging to the AppDevice or DeviceTemplate. For static configuration properties, this will include just the main interface.

2 lcaCpAppliesToLonMarkObject

The configuration property applies to one or more LonMarkObjects.

3 lcaCpAppliesToNetworkVariable

The configuration property applies to one or more NetworkVariable objects.

cpObject — The ConfigProperty object to be acted on.</td></tr>
<tr><td>Data Type</td><td>Short.</td></tr>
<tr><td>Read/Write</td><td>Read only.</td></tr>
<tr><td>Added to API</td><td>OpenLNS.</td></tr>
</table>

ByteLength

Summary	Returns the length (in bytes) of a single configuration property element. Multiplying this length by the number returned by the *Dimension* property gives you the total length (in bytes) of this configuration property. If this configuration property is not an array, the *Dimension* property returns 1, and the *ByteLength* property returns the full length of the configuration property. **Note**: If a configuration property inherits its type from a network variable that supports changeable types, and the type of that network variable is changed, this property will not be updated to reflect the change. You can check if the configuration property inherits its type by reading the *TypeInherits* property.
Availability	Local, full, and lightweight clients.
Syntax	*length* = *cpObject*.**ByteLength** **Element** — **Description** *cpObject* — The *ConfigProperty* object to be acted on *length* — The byte length of this *ConfigProperty* object
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ConfigProperty* object in the *ConstClassIds* constant: **26 lcaClassIdConfigProp** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

ConfigFileOffset

Summary	Specifies the byte offset into the configuration value file that implements the *ConfigProperty* object. A *ConfigProperty* object is implemented by a configuration network variable or a configuration value file. To determine how the configuration property is implemented, read the *ImplementationType* property of the *ConfigProperty* object. • If the configuration property is implemented by a configuration value file, this property identifies the byte offset into the configuration value file. To determine the value file to which this offset applies, read the *ConfigFileType* property. If the configuration property is an array implemented by a configuration value file, elements will occupy consecutive spaces in the file. This property returns the byte offset into the configuration value file of the first element. To determine the byte offset for subsequent elements, simply add the appropriate multiple of the element's length to this offset. To determine the length of the array's elements, see the *ByteLength* property. • If the configuration property is implemented by a configuration network variable, this property throws an **LCA, #165** exception *("configuration property not implemented by a configuration value file").* **Note**: This property is not currently available from within a *DeviceTemplate* object; therefore, it will throw a **LCA, #38 LCA_APP_DEVICE_REQUIRED** exception when accessed.
Availability	Local, full, and lightweight clients.
Syntax	*fileOffset* = *cpObject*.**ConfigFileOffset** **Element** — **Description** *fileOffset* — The byte offset into the configuration value file implementing this configuration property. To determine the value file to which this offset applies, read the *ConfigFileType* property of the *ConfigProperty* object. *cpObject* — The file-based *ConfigProperty* object to be acted on.
Data Type	Long.

Read/Write	Read only.
Added to API	OpenLNS.

ConfigFileType

Summary	Specifies the type of configuration value file that implements the *ConfigProperty* object. A *ConfigProperty* object is implemented by a configuration network variable or a configuration value file. To determine how the configuration property is implemented, read the *ImplementationType* property of the *ConfigProperty* object. • If the configuration property is implemented by a configuration value file, this property identifies the type of that value file. To determine the appropriate location within this value file, read the *ConfigFileOffset* property. • If the configuration property is implemented by a configuration NV, this property throws an **LCA, #165** exception *("configuration property not implemented by a configuration value file").* **Note**: This property is not currently available from within a *DeviceTemplate* object; therefore, it will throw a **LCA, #38 LCA_APP_DEVICE_REQUIRED** exception when accessed.
Availability	Local, full, and lightweight clients.

Syntax	*fileType* = *cpObject*.**ConfigFileType** **Element** — **Description** *fileType* — The type of the configuration value file implementing this configuration property. The possible values for this property, which are stored in the *ConstConfigFileTypes* constant, are as follows: **1 lcaConfigFileWritable** The configuration property is stored in the writable value file, which is the mandatory (first) value file that is typically stored in the read-write memory on the device. **2 lcaConfigFileConstant** The configuration property is stored in the constant value file, which is an optional (second) value file that is typically stored in the read-only memory on the device. *cpObject* — The network variable-based *ConfigProperty* object to be acted on.
Data Type	Short.
Read/Write	Read only.
Added to API	OpenLNS.

ConfigNv

Summary	If the configuration property is implemented by a network variable, returns the configuration network variable object. A *ConfigProperty* object is implemented by a configuration network variable or a configuration value file. To determine how the configuration property is implemented, read the *ImplementationType* property of the *ConfigProperty* object. • If the configuration property is implemented by a configuration network variable, this property returns that configuration network variable. To identify just the index of the configuration network variable, read the *ConfigNvIndex* property. The configuration network variable returned by this property refers back to this network variable-based configuration property using the *ImplementsCP* property of the *NetworkVariable* object. If the configuration property is an array that is

	implemented by an array of network variables, array elements will be implemented by consecutive network variables. This property returns the network variable that implements the first element. To retrieve the network variables for subsequent elements, increment the index of the network variable the appropriate number of times, and call the *ItemByIndex* method on the *NetworkVariables* property of the main *Interface* object. • If the configuration property is implemented by a configuration value file, this property throws an **LCA, #164** exception *("configuration property not implemented by a configuration network variable").* **Note**: This property is not currently available from within a *DeviceTemplate* object; therefore, it will throw a **LCA, #38 LCA_APP_DEVICE_REQUIRED** exception when accessed.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *cpObject*.**ConfigNv** **Element** — **Description** *nvObject* — The configuration network variable implementing this configuration property. *cpObject* — The network variable-based *ConfigProperty* object to be acted on.
Data Type	*NetworkVariable* object.
Read/Write	Read only.
Added to API	OpenLNS.

ConfigNvIndex

Summary	Specifies the index of the configuration network variable that implements the *ConfigProperty* object iguration property. A *ConfigProperty* object is implemented by a configuration network variable or a configuration value file. To determine how the configuration property is implemented, read the *ImplementationType* property of the *ConfigProperty* object. • If the configuration property is implemented by a configuration NV, this property specifies the index of that configuration network variable. • If the configuration property is implemented by a configuration value file, this property throws an **LCA, #164** exception *("configuration property not implemented by a configuration network variable").* To directly access the configuration network variable, read the **ConfigNv** property (read the NV's current selector). If the configuration property is an array that is implemented

	by an array of network variables, the array elements will be implemented by consecutive network variables. This property returns the network variable index of the first element. To determine the network variable index for subsequent elements, simply increment this network variable index the appropriate number of times. **Note**: This property is not currently available from within a *DeviceTemplate* object; therefore, it will throw a **LCA, #38 LCA_APP_DEVICE_REQUIRED** exception when accessed.
Availability	Local, full, and lightweight clients.
Syntax	*nvIndex* = *cpObject*.**ConfigNvIndex** **Element** — **Description** *nvIndex* — The index of the configuration network variable implementing this configuration property. *cpObject* — The network variable-based *ConfigProperty* object to be acted on.
Data Type	Short.
Read/Write	Read only.
Added to API	OpenLNS.

ConstantAttribute

Summary	Reads or writes the constant attribute of the configuration property. Many configuration properties contain a constant attribute, which indicates that the value of the configuration property should not be changed by a network management tool such as an OpenLNS application. You can use this property to set or turn off the constant attribute of a configuration property. Generally, configuration properties with the constant attribute set should not be modified by OpenLNS applications, or by any other means. However, version LNS Release 3.20 of the *LonMark Application-Layer Interoperability Guidelines* LNS Release 3.20 introduced constant, device-specific configuration properties, which can be modified by the device, but not by a network management tool. Modifiable device-specific configuration properties were introduced in version 3.4 of the *LonMark Application-Layer Interoperability Guidelines*. These device-specific configuration properties can be modified by the device, and by a network management tool. In the interim, many LonMark devices were defined with constant, device-specific configuration properties. Some of these configuration properties would more naturally have been defined as modifiable, device-specific configuration properties, if these had been defined by the LonMark

<table>
<tr><td></td><td>organization at the time. As a result, LNS 3.20 allows applications such as device plug-ins to modify the constant attribute by writing to this property. Set this property to True to turn on the constant attribute. Set this property to False to turn off the constant attribute.

When modifying this property, you should note that some configuration properties are stored in non-writable storage, such a ROM. In this case, OpenLNS cannot verify whether it is possible to update the value or not. If an OpenLNS application sets the constant attribute to False, and that configuration property's value is stored in non-writable memory, updating the value in the device will fail. The failure may or may not be reported to LNS and to the OpenLNS application, depending on how the value is stored. Even if the value is stored in modifiable memory, such as flash, it may be part of the checksumed application area. If this is the case, modifying the value may cause the device to detect an application checksum failure and enter the applicationless state. In addition, it is not possible for LNS to determine whether or not the device will operate correctly if the configuration property value is successfully modified in the device. For these reasons, you should take extreme care when modifiying this property from its initial setting, and only modify it if you have intimate knowledge of the device application, and are certain you can successfully update the value of the configuration property.

For more information on configuration property attributes, see the Configuration Property Flags section of the LonMark Application-Layer Interoperability Guidelines.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>returnValue = cpObject.ConstantAttribute

Element — Description

returnValue — A Boolean value indicating whether or not the constant attribute flag of the configuration property is set. If this property is set to True, then the attribute is set.

cpObject — The ConfigProperty object being acted on.</td></tr>
<tr><td>Data Type</td><td>Boolean.</td></tr>
<tr><td>Read/Write</td><td>This property is read-only for ConfigProperty objects that are accessed through AppDevice objects.

This property is read/write for ConfigProperty objects accessed through DeviceTemplate objects. Modifying this property for a ConfigProperty on a DeviceTemplate will affect all devices using that template.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.20.</td></tr>
</table>

Description

Summary	Stores description information about the *ConfigProperty* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *ConfigProperty* object. *object* — The *ConfigProperty* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

DeviceSpecificAttribute

Summary	Reads or writes the the device-specific attribute of the configuration property. Many configuration properties contain a device-specific attribute, which indicates that the value of the configuration property can be changed without the network manger's knowledge. For example, the configuration property value may be changed locally by the device itself, or remotely by another device on the network. As a consequence, the value of the device stored in the OpenLNS database may not be the most current value. It is not possible for LNS to know whether the value of a device-specific configuration property stored in the device or in the OpenLNS database is the current value. Depending on the setting of the *SourceOptions* property, the device-specific attribute affects how OpenLNS will read or write the value of the configuration property when it is accessed via a data point, or via the *ConfigProperty* object's *GetRawValuesEx* and *SetRawValuesEx* methods. It may also affect what configuration property values are downloaded or uploaded to the OpenLNS database when you call the *DownloadConfigProperties* and *UploadConfigProperties* methods. Initially, the device-specific attribute is defined by the programmatic interface of the device, in the device's external interface file or self-documentation. However, whether the configuration property can or should be modified without the knowledge of OpenLNS is often dependent on the

<table>
<tr><td></td><td>installation. Therefore, you can modify the device-specific attribute by writing to this property. Set this property to True to turn on the device-specific attribute. Set this property to False to turn it off.

For more information on configuration property attributes, see the Configuration Property Flags section of the LonMark Application-Layer Interoperability Guidelines.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>deviceSpecific = cpObject.DeviceSpecificAttribute

Element — Description
returnValue — A Boolean value indicating whether or not the device specific attribute flag of the configuration property is set. If this property is set to True, then the attribute is set.
cpObject — The ConfigProperty object being acted on.</td></tr>
<tr><td>Data Type</td><td>Boolean.</td></tr>
<tr><td>Read/Write</td><td>This property is read-only for ConfigProperty objects that are accessed through AppDevice objects.

This property is read/write for ConfigProperty objects accessed through DeviceTemplate objects. Modifying this property for a ConfigProperty on a DeviceTemplate will affect all devices using that template.

The LONMARK organization does not allow modifiable device-specific configuration properties to be stored in files accessed via FTP on devices that support only sequential access. Therefore, this property cannot be set to True on such a configuration property, and an attempt to do so will result in the NS, #293 lcaErrNsInvalidCpAttribute exception being thrown.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.20.</td></tr>
</table>

Dimension

Summary	Returns the length (in bytes) of a single configuration property element. Multiplying this length by the number returned by the *Dimension* property gives you the total length (in bytes) of this configuration property. If this configuration property is not an array, the *Dimension* property returns 1, and the *ByteLength* property returns the full length of the configuration property. **Note**: If a configuration property inherits its type from a network variable that supports changeable types, and the type of that network variable is changed, this property will not be updated to reflect the change. You can check if the configuration property inherits its type by reading the *TypeInherits* property.
Availability	Local, full, and lightweight clients.
Syntax	*length* = *cpObject*.**ByteLength** **Element** — **Description** *cpObject* — The *ConfigProperty* object to be acted on *length* — The byte length of this *ConfigProperty* object
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FlagsByte

Summary	Returns the flags byte defined for this configuration property. The flags byte represents the setting of several flags that specifiy the behavior of the configuration property, including whether it can be modified, and whether it is device-specific.
Availability	Local, full, and lightweight clients.
Syntax	*flagsByteValue* = *cpObject*.**FlagsByte** **Element** — **Description** *cpObject* — The *ConfigProperty* object to be acted on. *flagsByteValue* — The value of the FlagsByte property. The possible values for this element, which are stored in the *ConstCpFlags* constant, are as follows: **1 lcaCpFlagsDisabled** The configuration property is disabled. **2 lcaCpFlagsOffline**

	The configuration property is offline. **4 lcaCpFlagsConstant** The configuration property has the constant attribute set. You can use LNS to read or write the value of the constant attribute in some cases. See the *ConstantAttribute* and *DeviceSpecificAttribute* properties for more information. **8 lcaCpFlagsReset** The configuration property has been reset. **16 lcaCpFlagsManufactureOnly** The configuration property is a manufacturer-only configuration property. **32 lcaCpFlagsDeviceSpecific** The configuration property has the device-specific attribute set. You can use LNS to read or write the value of the device-specific attribute in some cases. See the *DeviceSpecificAttribute* property for more information.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FormatName

Summary	Specifies the formatting of the *ConfigProperty* object's *Value* property. The format is obtained on the computer running the OpenLNS application for Local client applications, and on the OpenLNS Server computer for Full and Lightweight client applications: OpenLNS will use a best-guess format to determine the default *FormatName* property. You can also write to this property to override the format determined by OpenLNS, **OpenLNS Best-Guess Format** OpenLNS will use a best-guess format when determining the default for this property, using the following procedure: 1. If the *ConfigProperty* object's type is a standard configuration property type (SCPT) or a user configuration property type (UCPT) that is found in the Device Resource File Type and Format files, and it is not

	a SNVT_xxx type, it will be formatted as specified by the format file. 2. If the *ConfigProperty* object's type does not appear in the Device Resource File Type and Format files, or if it appears but is designated as a SNVT_xxx type, the OpenLNS Server will check to see if it applies to a standard network variable. If it does, it will be formatted according to the SNVT format specification in STANDARD.FMT. 3. As a third option, OpenLNS will use the Hex Byte String format as the default. In this case, each byte of the raw value assigned to the *ConfigProperty* object will be displayed as two hexadecimal digits. For example, the byte sequence 1,2,3,4 would be displayed as 01020304. **Overriding the Best-Guess Format** When you successfully write to this property, the new value will be stored persistently in the OpenLNS database. However, you can revert the property back to the default value chosen by OpenLNS at any time by writing an empty string or the exact name of the type (e.g. SCPTmaxFlow) to it. There are several possible syntaxes you can use when writing to this property if you want to override the best-guess format: 1. A format name beginning with "SCPT" for standard type formats or "UCPT" for user-defined type formats (e.g., "SCPTmaxFlow"). The Data Server will use the resource file catalog specified using the *LdrfCatalogPath* property to search for SCPT and UCPT formats. There may be several different formats for the same network variable type. For example, the LonMaker Integration Tool has two alternate formats for the type SCPTmaxFlow, and those formats are named SCPTmaxFlow#SI and SCPTmaxFlow#US. In OpenLNS, the default formats for types such as SCPTmaxFlow are determined using the Windows regional settings of the computer where the data is formatted (on the computer housing the OpenLNS Server for local and lightweight clients, or on the computer running the application for full clients). To determine the settings being used on a computer, open the Windows Control Panel and double-click the Regional Options icon. Select the Numbers tab. OpenLNS uses the value of the Measurement System field (either U.S. or metric) on this tab to determine the default format to use for types such as SCPTmaxFlow. You can also specify a full format name (e.g. SCPTmaxFlow #SI or SCPTmaxFlow#US) to select a specific format for that type. For UNVTs, you must always specify a fully-qualified format name. 2. In some cases, a format exists for a given SCPT or UCPT

<table>
<tr><td></td><td>that has the same name as the underlying type. This is called the root format for the type, and may be different from the default format for that type. In order to explicitly use this root format, you will need to append the '#' character to the format name to indicate that you are specifying the format name (and not the type name) for this type. For example, if you read the value of a configuration property of type SCPTminSndT when the CategoryPreferenceList property of the FormatLocale object your application is using is set to LO, and you set the FormatName property to SCPTminSndT, the data stored in the Value property will be formatted using the SCPTminSndT#LO format. However, if you set the FormatName property to SCPTminSndT#, the Value property will be formatted using the root SCPTminSndT format.

3. A fully-qualified format name, expressed in the following syntax: "#<progID>[<scope>].<format name>"

In this syntax, the "#", "[", "]" and "." characters are literal characters. A hex byte string represents the program ID. The scope is a one-digit string. It represents a filter that indicates relevant parts of the program ID, and may be one of the following:

0 - Standard

1 - Device Class

2 - Device Class and Usage

3 - Manufacturer

4 - Manufacturer and Device Class

5 - Manufacturer, Device Class, and Device Subclass

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

For example, #800001128000000[4].UNVT_date_event

4. The name of one of the Data Server built-in types, which include "INT", "REAL", "DISCRETE", "BINARY", "RAW" and "RAW_HEX".

Note: Use of the old syntax of <code><file name>.<type name></code> for FormatName is no longer supported. This notation was used for pre-LONMARK resource files.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>formNameValue = cpObject.FormatName

Element — Description
cpObject — The ConfigProperty object to be acted on.
formNameValue — The FormatName as a string.</td></tr>
</table>

Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

FuncProfileDescription

Summary	Provides a desciptive comment of the functional profile associated with the *ConfigProperty* object. This property is accessed from the functional profile template file associated with the object. This property is language dependent. Set the *System* object's *ResourceLanguageId* to control the language.
Availability	Local, full, and lightweight clients.
Syntax	*description* = *object*.**FuncProfileDescription** **Element** — **Description** *object* — The *ConfigProperty* object to be acted on. *description* — The returned functional profile description string.
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

FuncProfileName

Summary	Returns the functional profile name associated with the *ConfigProperty* object. This property is accessed from the functional profile template file associated with the object. The name returned by this property is accessed from the functional profile template file associated with this object. This property is language dependent. Set the *System* object's *ResourceLanguageId* to control the language.
Availability	Local, full, and lightweight clients.
Syntax	*typeNameValue* = *object*.**FuncProfileName** **Element** — **Description** *object* — The *ConfigProperty* object to be acted on. *typeNameValue* — The functional profile name to be returned.
Data Type	String.

Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

FuncProfileProgrammaticName

Summary	Returns the functional profile programmatic name associated with the *ConfigProperty* object. This name is accessed from the functional profile template file associated with the object. The programmatic name is the base name stored for the object; it is not language dependent like the *FuncProfileName* property.
Availability	Local, full, and lightweight clients.
Syntax	*progNameValue* = *Object*.**FuncProfileProgrammaticName** **Element** — **Description** *progName* — The functional profile programmatic name of the object. *Object* — The *ConfigProperty* object to be acted on.
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *ConfigProperty* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the object. *object* — The object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ImplementationType

Summary	Specifies whether the *ConfigProperty* object is implemented by a configuration network variable or a configuration value file.

<table>
<tr><td></td><td>Once the implementation type is known, a client can retrieve the appropriate addressing information for the configuration property by reading either the ConfigNvIndex property (for NV-based CPs), or the ConfigFileType and ConfigFileOffset properties (for file-based CPs).

This information is useful for tools that need to access configuration property values when OpenLNS is not available. For example, the SmartServer needs this information to manage configuration property values when it is running in standalone mode.

Notes: Modifying configuration property values without OpenLNS is typically nor recoemmnded because it causes the application device and the OpenLNS database to lose synchronization.

This property is not currently available from within a DeviceTemplate object; therefore, it will throw a LCA, #38 LCA_APP_DEVICE_REQUIRED exception when accessed.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>cpType = cpObject.ImplementationType

Element — Description

cpType — The configuration property's implementation type.

The possible values for this property, which are stored in the ConstImplementationTypes constant, are as follows:

1 lcaImplementedByNetworkVariable

The configuration property is implemented by a configuration network variable. To determine the index of the configuration variable, read the ConfigNvIndex property of the ConfigProperty object.

2 lcaImplementedByConfigFile

The configuration property is implemented by a configuration value file. To determine the type and offset into the configuration value file, read the ConfigFileType and ConfigFileOffset properties of the ConfigProperty object.

cpObject — The ConfigProperty object to be acted on.</td></tr>
<tr><td>Data Type</td><td>Short.</td></tr>
<tr><td>Read/Write</td><td>Read only.</td></tr>
</table>

Added to API	OpenLNS.

Mode

Summary	Returns the scope of the LonMark resource file containing the definition of the type used by this configuration property. The value is acquired from the device during the import of the LonMark information during installation.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classMode* = *cpObject*.**Mode** **Element** — **Description** *classMode* — The scope of the LonMark resource file containing the definition of the type this configuration property is using. The possible values for this property, which are stored in the *ConstResourceScope* constant (and in the *ConstConfigPropertyModes* constant for LNS versions prior to LNS Release 3.20), are as follows: **0 lcaResourceScopeStandard** Applies to all devices. **1 lcaResourceScopeClass** Applies to all devices of a specified device class from any manufacturer. **2 lcaResourceScopeSubclass** Applies to all devices of a specified device class and device subclass from any manufacturer. **3 lcaResourceScopeMfg** Applies to all devices of a specified manufacturer. **4 lcaResourceScopeMfgClass** Applies to all devices of a specified manufacturer and device class. **5 lcaResourceScopeMfgSubClass** Applies to all devices of a specified manufacturer, device class and device subclass. **6 lcaResourceScopeMfgModel** Applies to all devices of a specified manufacturer, device class, device

		subclass and model. **-1 lcaResourceScopeUnknown** The scope of the resource file is not known, or could not be found. *cpObject* The *ConfigProperty* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.	
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.	
Syntax	*stringValue* = *object*.**Name** **Element** **Description** *stringValue* The name of the object. *object* The object to be acted on.	
Data Type	String.	
Read/Write	Read/write.	
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.	

Parent

Summary	Returns the object that spawned the current child object.	
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.	
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.	

Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

Precision

Summary	Determines the precision of the *ConfigProperty* object. This represents the number of digits that will be used when the value of the configuration property is read and displayed. If the data type used by the configuration property is a single float type, the property defaults to 6 and has a range of 0-7. If the data type used by the configuration property is a double float type, the property defaults to 15 and has a range of 0-17.
Availability	Local, full, lightweight, and independent clients.
Syntax	*precisionValue* = *object*.**Precision** **Element** — **Description** *precisionValue* — The precision of the configuration property value. *object* — The *ConfigProperty* object to be acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

TypeInherits

Summary	Indicates whether the configuration property inherits its type from the network variable to which it applies.
Availability	Local, full, lightweight, and independent clients.
Syntax	*inherits* = *configProperty*.*TypeInherits* **Element** — **Description** *inherits* — The value of this property is taken from the definition of the configuration property in the device resource files. If this property is set to 1, then the configuration property always inherits its type from the network variable to which it applies. In this case, you may need to program your application to

		account for changes to the configuration property's type. Only configuration properties that apply to network variables can inherit types. For example, consider a configuration property with the *TypeInherits* property set to 1. If an application changes the type of the network variable to which the configuration property belongs, then OpenLNS changes the type of the configuration property automatically, since the *TypeInherits* property is set to 1. You would need to know about this change when reading the value of the configuration property, and you can use this property to keep track of which configuration properties may be modified by OpenLNS in this fashion.
	configProperty	The *ConfigProperty* object being acted upon.
Data Type	Short.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

TypeIndex

Summary	Returns the type index of the configuration property. Each *ConfigProperty* object uses a type defined in the standard LonMark resource files, or in a user-defined resource file. This property returns the index assigned to that type within the resource file containing its definition. If the *Mode* property is set to **lcaResourceScopeStandard (0)**, then the *ConfigProperty* object's type is defined in the standard resource files.	
Availability	Local, full, lightweight, and independent clients.	
Syntax	*typeIndexValue* = *cpObject*.**TypeIndex**	
	Element	**Description**
	cpObject	The *ConfigProperty* object to be acted on.
	typeIndexValue	The type index for the specified *ConfigProperty* object.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ValueStatus

Summary	Determines the current state of the value stored in the OpenLNS database for the *ConfigProperty* object. In some cases, the stored value for a configuration property in the OpenLNS database may not match the actual value of the configuration property in the physical device on the network. You can use this property to determine the current state of the value stored in the OpenLNS database for the configuration property.
Availability	Local, full, lightweight, and independent clients.
Syntax	*returnValue* = *cpObject*.**ValueStatus** **Element** — **Description** *returnValue* — The current status of the Value and RawValue properties. The values for this property, which are stored in the *ConstConfigPropertyValueMgmtStatus* constant, are as follows: **0 lcaConfigPropertyValueMgmtStatusUnknown** The configuration property value is not currently stored in the OpenLNS database. To resolve this, follow these steps: 1. Call the *GetDataPoint* method to create a *DataPoint* for the configuration property. Use the **lcaDataSourceOptionsFromDevice (1)** value as the *options* element when you create the data point. This enables you to read the value of the configuration property from the device when the data point is created. 2. Set the *SourceOptions* property of the newly created *DataPoint* to **lcaDataSourceOptionsDatabaseOnly (2)**. 3. Call the *Write* method to set the value from the device in the OpenLNS database. Alternatively, you can load all unknown configuration property values for the device into the

		OpenLNS database by calling the *UploadConfigProperties* method, and specifying **lcaConfigPropOptLoadUnknown (4)** as the *options* element. **1 lcaConfigPropertyValueMgmtStatusPendingUpdate** The configuration property value is stored in the OpenLNS database, and has been updated. This value needs to be written to the device. OpenLNS will write the value to the device automatically as part of its normal sweep operations. **2 lcaConfigPropertyValueMgmtStatusNormal** The configuration property value is stored in the device, and there are no pending updates. Typically this means that the values of the configuration property in the device, and in the OpenLNS database, are identical. However, if the value of the configuration property can be modified outside of LNS (for example, it is a device-specific configuration property), the value in the database and in the device may be different—even if the *ValueStatus* property is set to this value.
	cpObject	The *ConfigProperty* object being acted on.
Data Type	Long	
Read/Write	Read only.	
Added to API	LNS Release 3.2.	

ConnectDescTemplate

A *ConnectDescTemplate* object represents a generic connection description template that can be applied to multiple network variable *connections*. This template describes the attributes for a connection, including the messaging service. Using this object, standard connection description templates can be defined and associated with multiple connections.

The values you assign to the active attributes of a connection description template will be used by all of the devices added to the connections associated with that description template. You can activate the attributes of the connection description template with the *PropertyOptions* property.

Description	A generic connection description template that can be applied to multiple network variable *connections*.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*ConnectDescTemplates* collection object. *NetworkVariable* object.
Default Property	*Name*
Methods	None
Properties	• *AliasOptions* • *BroadcastOptions* • *ClassId* • *Description* • *Handle* • *MTHubs* • *Name* • *NVHubs* • *Parent* • *PropertyOptions* • *ReceiveTimer* • *RepeatCount* • *RepeatTimer* • *RetryCount* • *ServiceType* • *TransmitTimer* • *UseAuthenticationFlag* • *UsePriorityFlag*

Methods

The *ConnectDescTemplate* object does not contain any methods.

Properties

The *ConnectDescTemplate* object contains the following properties:

- *AliasOptions*
- *BroadcastOptions*
- *ClassId*
- *Description*
- *Handle*
- *MTHubs*
- *Name*
- *NVHubs*
- *Parent*
- *PropertyOptions*
- *ReceiveTimer*
- *RepeatCount*
- *RepeatTimer*
- *RetryCount*
- *ServiceType*
- *TransmitTimer*
- *UseAuthenticationFlag*

- *UsePriorityFlag*

AliasOptions

Summary	Specifies how network variable aliases should be used in the connection. OpenLNS determines how it will use aliasing for a network variable in a connection when the network variable is added to the connection. As a result, changing the value of this property in the *ConnectDescTemplate* object being used by a connection will not affect the status of network variables that have already been added to the connection. It will only affect how aliasing will be used for network variables that are added to the connection after the *AliasOptions* property has been modified. If you need to change the aliasing options for network variables that have already been added to a connection, it will be necessary to re-build the connection. You can do this following these steps: 1. Call the *Disconnect* method on the connection's hub network variable. 2. Set the property of the hub network variable to the desired value. 3. Add the target network variables back to the connection with the *AddTarget* method. 4. Call the *Connect* method on the hub network variable.
Availability	Local, full, and lightweight clients.
Syntax	*optionsValue* = *cdtObject*.**AliasOptions** **Element** — **Description** *cdtObject* — The *ConnectDescTemplate* object to be acted on. *optionsValue* — The desired alias options.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

BroadcastOptions

Summary	Specifies the broadcast options for this *ConnectDescTemplate* object.
Availability	Local, full, and lightweight clients.
Syntax	*optionsValue* = *cdtObject*.**BroadcastOptions** **Element** — **Description** *optionsValue* — The broadcast options. The valid values for this element, which are contained in

		the *ConstConnBroadcastOptions* constant, are as follows: **0 lcaBroadcastNever** No broadcast addressing will be used. **1 lcaBroadcastGroup** If a group address is required, but no groups are available, broadcast addressing will be used. A subnet broadcast will be used if possible; otherwise, a domain broadcast will be used. **2 lcaBroadcastAlways** Broadcast addressing will be used in all cases. A subnet broadcast will be used, if possible; otherwise, a domain broadcast will be used.
	cdtObject	The *ConnectDescTemplate* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

ClassId

Summary	Identifies the object class of this object.	
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.	
Syntax	*classIdValue* = *object*.**ClassId**	
	Element	**Description**
	classIdValue	The object class of the object. The following value is defined for the *ConnectDescTemplate* object in the *ConstClassIds* constant: **42 lcaClassIdConnectDescTemplate**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	
Comments	This property allows the object type to be determined when it	

	is unknown (for example, when the object was accessed using the *Parent* property of another object).

Description

Summary	Stores description information about the *ConnectDescTemplate* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*updateType* = *appDeviceObject*.**ConnectionUpdateType** **Element** — **Description** *stringValue* — A string description of the *ConnectDescTemplate* object. *object* — The *ConnectDescTemplate* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Handle

Summary	Contains the handle associated with the *ConnectDescTemplate* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property and may also be read using the *Index* property.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the object. *object* — The object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MTHubs

Summary	Returns a collection of Message Tags that have connections that use this connection description template. When a new hub is added, it may not be added to the end of

	the list of hubs; therefore, you should update the cached copy of the complete hub list when you add or delete a hub.
Availability	Local, full, and lightweight clients.
Syntax	*mtCollection* = *cdtObject*.**MTHubs** **Element** — **Description** *mtCollection* — The returned *MessageTags* collection. *cdtObject* — The specified *ConnectDescTemplate* object.
Data Type	*MessageTags* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NVHubs

Summary	Returns a collection of Network Variables that have connections that use this Connection Description Template. When a new hub is added, it may not be added to the end of the list of hubs; therefore, you should update the cached copy of the complete hub list when you add or delete a hub.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.

Syntax	*nvColl* = *cdtObject*.**NVHubs** **Element** — **Description** *cdtObject* — The *ConnectDescTemplate* object to be acted on. *nvColl* — The collection of *NetworkVariables* to be returned.
Data Type	*NetworkVariables* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

PropertyOptions

Summary	Specifies which attributes in the connection description template are active. If an attribute is not specified as active, the attributes in the network variable declarations (in the device's application program) or the Object Server's internal defaults are used in connections associated with the template. If an attribute is designated as active, the corresponding value supplied in the *ConnectDescTemplate* object overrrides the default.
Availability	Local, full, and lightweight clients.

Syntax	*propOptionsValue* = *cdtObject*.**PropertyOptions**

Element	Description
cdtObject	The *ConnectDescTemplate* object to be acted on.
propOptionsValue	The value indicating which attributes are active. Multiple options can be ORed together to form the set of attributes that will be considered active in the connection. For example, you could OR **lcaConnPropsServiceType** with **lcaConnPropsPriority** to activate the *ServiceType* and *UsePriorityFlag* properties. The values of the *ServiceType* and *UsePriorityFlag* properties in the devices used in the connection would then be overridden by the values stored in the *ConnectDescTemplate* object for those properties. The values for all other properties would still be taken from device and internal defaults. The possible options for this element, which are contained in the *ConstConnPropOptions* constant, are as follows: **0 lcaConnPropsNone** **1 lcaConnPropsServiceType** Select this value to activate the *ServiceType* property of the *ConnectDescTemplate* object. **2 lcaConnPropsPriority** Select this value to activate the *UsePriorityFlag* property of the *ConnectDescTemplate* object. **4 lcaConnPropsAuth** Select this value to activate the *UseAuthenticationFlag* property of the *ConnectDescTemplate* object. **8 lcaConnPropsRetryCount** Select this value to activate the *RetryCount* property of the *ConnectDescTemplate* object. **16 lcaConnPropsRepeatCount** Select this value to activate the

		RepeatCount property of the *ConnectDescTemplate* object. **32 lcaConnPropsRepeatTimer** Select this value to activate the *RepeatTimer* property of the *ConnectDescTemplate* object. **64 lcaConnPropsReceiveTimer** Select this value to activate the *ReceiveTimer* property of the *ConnectDescTemplate* object. **128 lcaConnPropsTransmitTimer** Select this value to activate the *TransmitTimer* property of the *ConnectDescTemplate* object. **256 lcaConnPropsSuppressSourceAddr** This option is no longer supported.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

ReceiveTimer

Summary	Determines the time within which a received message with a duplicate reference ID will be considered a duplicate message. This property will only be used by the Object Server if the **lcaConnPropsReceiveTimer** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property.	
Availability	Local, full, and lightweight clients.	
Syntax	*receiveTimerValue* = *cdtObject*.**ReceiveTimer**	
	Element	**Description**
	cdtObject	The *ConnectDescTemplate* object to be acted on.
	receiveTimerValue	The encoded value of the receive timer. This value determines the time period within which receiving devices will treat messages from the same source with the same reference ID as duplicate messages. If the **lcaConnPropsReceiveTimer** option is not activated in the *PropertyOptions* property, the

	default receive timer is calculated based on the network topology, channel delays, and other connections that exist on the system. This property accepts a range of encoded values from 0 to 15. The encoded receive timer values are as follows: **Encoded Value** / **Seconds** 0 / 0.128 1 / 0.192 2 / 0.256 3 / 0.384 4 / 0.512 5 / 0.768 6 / 1.024 7 / 1.536 8 / 2.048 9 / 3.072 10 / 4.096 11 / 6.144 12 / 8.192 13 / 12.288 14 / 16.384 15 / 24.576
Data Type	Integer.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

RepeatCount

Summary	Indicates the number of times to repeat a message sent using the repeated message service (if the *ConnectDescTemplate* is using the repeated message service, which is equal to **lcaSvcUnackdRpt**). This property will only be implemented if you set the **lcaConnPropsRepeatCount** option in the *ConnectDescTemplate* object's *PropertyOptions* property. If this property is not activated by the *PropertyOptions* property, the default value is calculated based on network topology. You can set the frequency in which the repeat messages will be sent by writing to the *RepeatTimer* property of the *ConnectDescTemplate* object.
Availability	Local, full, and lightweight clients.

Syntax	*repeatCountValue*= *cdtObject*.**RepeatCount** **Element** **Description** *cdtObject* The *ConnectDescTemplate* object to be acted on. *repeatCountValue* The number of times (0–15) the messages will be repeated. This property is used to establish the repeat count.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

RepeatTimer

Summary	The encoded value of the repeat timer used for network management messages. This value defines the interval to wait between repeated messages sent using the unacknowledged/repeat message service during network management operations. You must set the **lcaConnPropsRepeatTimer** option in the *ConnectDescTemplate* object's *PropertyOptions* property to implement the repeat timer. If the option is not set in *PropertyOptions*, then the connection will use the default value for the property, which is based on network topology. You can set the number of repeat messages that will be sent by writing to the *RepeatCount* property of the *ConnectDescTemplate* object. **Note**: The value of this property in a *ConnectDescTemplate* applies to all permanent monitor points involved in the connections that are using that template. However, to set the repeat timer value for monitor and control messages for temporary monitor points, you need to use the *DsRepeatTimer* property.
Availability	Local, full, and lightweight clients.
Syntax	*repeatTimerValue* = *object.RepeatTimer* **Element** **Description**

<table>
<tr><td></td><td>repeatTimerValue</td><td>The encoded value of the repeat timer.

The default value for this property is determined based on the network topology. It is recommended that you do not change this from the default value. If the default value is not suitable for your application, it is recommended that you use the Delay property of each Channel object on the network to ensure that each message is sent at the correct interval.

However, this property does accept a range of encoded values from 0 to 15. The encoded repeat timer values are as follows:

Encoded Value Seconds
0 0.016
1 0.024
2 0.032
3 0.048
4 0.064
5 0.096
6 0.128
7 0.192
8 0.256
9 0.384
10 0.512
11 0.768
12 1.024
13 1.536
14 2.048
15 3.072

You can also write the value 254 to the property at any time to restore it to the default.

If you assign this property a value outside the acceptable range, the NS, #29 lcaErrNsOutOfRange exception will be thrown.

object The ConnectDescTemplate to be acted on.</td></tr>
<tr><td>Data Type</td><td colspan="2">Integer.</td></tr>
<tr><td>Read/Write</td><td colspan="2">Read/write.</td></tr>
<tr><td>Added to API</td><td colspan="2">Prior to LNS Release 3.0.</td></tr>
</table>

RetryCount

Summary	Sets the maximum number of times to resend messages sent with the request/response, unacknowledged/repeat, or acknowledged messaging services. This also sets the retry count for polling. You can set the service type for a connection by writing to the *ServiceType* property of the *ConnectDescTemplate* object. • If the acknowledged (**lcaSvcAckd)** or request/response (**lcaSvcRequest**) service types are selected, the retry count sets the number of retries to send when no acknowledgment of (or response to) the original message is received. You can set the interval for these retries by writing to the *TransmitTimer* property of the *ConnectDescTemplate* object. • If the unacknowledged/repeat (**lcaSvcUnackdRpt)** service type is selected, the retry count sets the number of repeats to send for each network management message. You can set the interval for the repeat messages by writing to the *RepeatTimer* property of the *ConnectDescTemplate* object. **Note**: In the *ConnectDescTemplate* object, the value of this property applies to all permanent monitor and control messages sent by connections using that template. However, to set the retry count for monitor and control messages for temporary monitor sets, you need to use the *DsRetryCount* property of the System object.
Availability	Local, full, and lightweight clients.
Syntax	*retryCountValue* **=** *object*.**RetryCount** **Element** — **Description** *retryCountValue* — The retry count value. The default value is calculated based on network topology, and the property has a range of 0-15. This property will only be used for a *ConnectDescTemplate* object if the **lcaConnPropsRetryCount** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property. *object* — The *ConnectDescTemplate* or System object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ServiceType

Summary	Stores the messaging service to be used for the connection. **Note**: This property will only be read by the OpenLNS Object Server if the **lcaConnPropsServiceType** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property. If the **lcaConnPropServiceType** option is not set, then OpenLNS Object Server will determine the service type for connections using the *ConnectDescTemplate* object by reading the *ServiceType* property of the connection's hub network variable.
Availability	Local, full, and lightweight clients.
Syntax	*serviceTypeValue* = *object*.**ServiceType** **Element** — **Description** *serviceTypeValue* — The service type to be used. The enumerated values for this element, which are contained in the *ConstServiceTypes* constant, are as follows: **0 lcaSvcAckd** Acknowledged messaging service. The device sends an acknowledgment message after it has received the message. If your application will be sending messages to large numbers of devices at once, one of the unacknowledged messaging services may be desirable, as the acknowledgment messages may generate a significant amount of network traffic. **1 lcaSvcUnackdRpt** Unacknowledged repeat messaging service. The device does not send acknowledgment messages; however, repeat messages are sent to the device after the initial message is sent to it to ensure that it reaches its destination. You can set the number of repeat messages to send, and the interval at which they will be sent, by writing to the *RepeatCount* and *RepeatTimer* properties. **2 lcaSvcUnackd** Unacknowledged messaging service. The device does not send

		acknowledgment messages. Do not use this service type on channels that support altnerate frequencies because the message will only be sent using the primary path. See the *AltPathType* property for more information. **3 lcaSvcRequest** Request/Response messaging service. You can use this value when sending explicit messages if the device receiving the message is designed to send a response message for the specified message code.
	object	The *ConnectDescTemplate* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

TransmitTimer

Summary	Specifies the interval between **lcaSvcAckd** and **lcaSvcRequest** messages.	
Availability	Local, full, and lightweight clients.	
Syntax	*transTimerValue = cdtObject.TransmitTimer*	
	Element	**Description**
	cdtObject	The *ConnectDescTemplate* object to be acted on.
	transTimerValue	The value of the transmit timer. This property will only be read by the Object Server if the **lcaConnPropsTansmitTimer (128)** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property. Otherwise, the transmit timer used by the connection will be calculated automatically based on the network topology and channel delays. This property accepts a range of encoded values from 0 to 15. The encoded transmit timer values are as follows: **Encoded Value** **Seconds** 0 0.016

	1	0.024
	2	0.032
	3	0.048
	4	0.064
	5	0.096
	6	0.128
	7	0.192
	8	0.256
	9	0.384
	10	0.512
	11	0.768
	12	1.024
	13	1.536
	14	2.048
	15	3.072
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

UseAuthenticationFlag

Summary	Indicates whether this connection uses authentication.
Availability	Local, full, and lightweight clients.
Syntax	*useAuthFlag* = *cdtObject*.**UseAuthenticationFlag** **Element** — **Description** *cdtObject* — The *ConnectDescTemplate* object to be acted on. *useAuthFlag* — A Boolean value. This property will only be read by the Object Server if the **lcaConnPropsAuth** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property. • A True value indicates that authentication is enabled for this connection. • A False value indicates that authentication is disabled for this connection. • If this value is not explicitly set for a *ConnectDescTemplate* object, the authentication settings for connections using the template will default to the setting of the hub network variable's

	AuthenticationFlag. This property is used for network variable connections only.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

UsePriorityFlag

Summary	Indicates whether this connection uses priority messaging. With *priority messaging*, the device with the highest priority sends its packet before any other devices can send theirs. This is accomplished by assigning each priority device a time slot where it can transmit before all other lower priority and non-priority devices. These time slots consume network bandwidth; therefore, priority messaging should only be used for critical devices and data.
Availability	Local, full, and lightweight clients.
Syntax	*usePriorityFlag* = *cdtObject*.**UsePriorityFlag** **Element** — **Description** *cdtObject* — The *ConnectDescTemplate* object to be acted on. *usePriorityFlag* — A Boolean value. This property will only be read by the Object Server if the **lcaConnPropsPriority** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property. • A True value indicates that this connection uses priority messaging. • A False value indicates that this connection does not use priority messaging. • If this value is not explicitly set for a *ConnectDescTemplate* object, it defaults to the network variable's *DsPriority* property.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ConnectDescTemplates

A *ConnectDescTemplates* object is a collection of *ConnectDescTemplate* objects. The instance of this object is accessed through the *TemplateLibrary* object holds all of the *ConnectDescTemplate* objects in the system. The following table summarizes the *ConnectDescTemplates* object.

Description	A collection of *ConnectDescTemplate* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*TemplateLibrary* object.
Default Property	*Item*
Methods	• *Add* • *ItemByHandle* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *ConnectDescTemplates* object contains the following methods.

- *Add*
- *ItemByHandle*
- *Remove*

Add

Summary	Defines a new *ConnectDescTemplate* object. A *ConnectDescTemplate* object needs to be created and initialized before use. It may optionally be assigned to a *NetworkVariable* or *MessageTag* object before invoking the *Connect* method on that *NetworkVariable* or *MessageTag* object.
Availability	Local, full, and lightweight clients.
Syntax	*cdTemplateObject* = *cdTemplatesColl*.**Add**(*cdTemplateName*) **Element** — **Description** *cdTemplateObject* — The newly defined *ConnectDescTemplate* object. *cdTemplatesColl* — The *ConnectDescTemplates* collection object. *cdTemplateName* — A String containing the name of the connection description template.
Added to API	Prior to LNS Release 3.0.

ItemByHandle

Summary	Retrieves a *ConnectDescTemplate* object, specified by its handle, from a *ConnectDescTemplates* collection. The *ConnectDescTemplate* object to be retrieved must be specified by its handle.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.**ItemByHandle**(*handle*) **Element** — **Description** *object* — The object retrieved from the collection. *coll* — The collection object. *handle* — A Long value specifying the handle of the object to be retrieved.
Added to API	LNS Release 3.0.
Added to API	Prior to LNS Release 3.0.

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed. *name* — A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *ConnectDescTemplates* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to

	Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ComponentApps* object in the *ConstClassIds* constant: **43 lcaClassIdConnectDescTemplates** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *ConnectDescTemplate* object from a *ConnectDescTemplates* collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *ConnectDescTemplate* object with the *Name* property by

	passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The object retrieved from the collection. *collObject* — The collection object to be acted on. *index* — A Long type specifying the ordinal index of the object to retrieve. *stringExpression* — A string type specifying the name of the object to retrieve.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using

	For Each ... Next statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

Connections

A *Connections* object contains all of the network variable and message tag connections in the system. Each connection is represented by the connection hub network variable or message tag. You can create connections using the *AddTarget* and *Connect* methods.

The following table summarizes the *Connections* object.

Description	Connects all of the network variable and message tag connections in the system.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None
Methods	None
Properties	• *ClassId* • *MessageTags* • *NetworkVariables* • *Parent*

Methods

The *Connections* object does not contain any methods.

Properties

The *Connections* object contains the following properties:

- *ClassId*
- *MessageTags*
- *NetworkVariables*
- *Parent*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Connections* object in the *ConstClassIds* constant: **18 lcaClassIdConnections** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).

MessageTags

Summary	Returns the *MessageTags* collection associated with the object.
Availability	Local, full, and lightweight clients.
Syntax	*mtCollection* = *object*.**MessageTags** **Element** — **Description** *mtCollection* — The returned *MessageTags* collection. *object* — The *Connections* object to be acted on

Data Type	*MessageTags* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NetworkVariables

Summary	Represents all of the network variable connections in the system with a *NetworkVariables* collection object that contains the hub network variables for all connections.
Availability	Local, full, and lightweight clients.
Syntax	*nvCollection* = *object*.**NetworkVariables** **Element** — **Description** *nvCollection* — The returned *NetworkVariables* collection. *object* — The *Connections* object to be acted on
Data Type	*NetworkVariables* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

CreditInfo

OpenLNS applications can determine a system's current licensing status by reading the properties of the *CreditInfo* object. These properties return information indicating the number of days until the LNS activation license expires or the number of application devices defined in a system.

Note: As of OpenLNS, the *DeficitCredits*, *LicensedCredits*, *LicenseType*, *MaxDeficitCredits*, and *UsedCredits* properties of the *CreditInfo* object have been deprecated and no longer provide any useful information.

The following table summarizes the *CreditInfo* object.

Description	A single point of data in a LONWORKS network.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System*.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *DaysRemaining* • *DeficitCredits* • *LicensedCredits* • *LicenseType* • *MaxDeficitCredits* • *UsedCredits*

Methods

The *CreditInfo* object does not contain any methods:

Properties

The *CreditInfo* object contains the following properties:

- *ClassId*
- *DaysRemaining*
- *DeficitCredits*
- *LicensedCredits*
- *LicenseType*
- *MaxDeficitCredits*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *CreditInfo* object in the *ConstClassIds* constant: **53 lcaClassIdCreditInfo** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

DaysRemaining

Summary	Returns the number of days until the OpenLNS activation license expires or 255, whichever is lesser. **Note**: In previous LNS releases, this property indicated the number of days remaining before the OpenLNS Server license would expire due to being in deficit mode. When not in deficit mode, the value was always 255.
Availability	Local, full, and lightweight clients.
Syntax	*numDays* = *object*.**DaysRemaining** **Element** **Description** *numDays* The number of days until the LNS activation license expires. *object* The *CreditInfo* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DeficitCredits (Deprecated)

Summary	Returns **0**. **Note**: In previous LNS releases, this property indicated how many credits had been consumed over the number of credits purchased.
Availability	Local, full, and lightweight clients.
Syntax	*deficitValue* = *creditObject*.**DeficitCredits** **Element** **Description** *deficitValue* The number of deficit credits, which is

	always **0**. *creditObject* The *CreditInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LicensedCredits (Deprecated)

Summary	Returns the currently licensed capacity, which is typically either **6** for trial licenses, or **65,535** for non-trial licenses. **Note**: In previous LNS releases, this property indicated the number of credits defined in the LNS license.
Availability	Local, full, and lightweight clients.
Syntax	*licValue* = *object*.**LicensedCredits** **Element** **Description** *licValue* The number of LNS Device Credits. *creditObject* The *CreditInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LicenseType (Deprecated)

Summary	Returns **lcaLNSLicenseTypeNone**. **Note**: In previous LNS releases, this property indicted the type of license (either **lcaLNSLicenseTypeNone** or **lcaLNSLicenseTypeStandard**).
Availability	Local, full, and lightweight clients.
Syntax	*licValue* = *object*.**LicensedCredits** **Element** **Description** *licValue* The number of LNS Device Credits. *creditObject* The *CreditInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MaxDeficitCredits (Deprecated)

Summary	Returns **0**. **Note**: In previous LNS releases, this property indicted the the maximum number of deficit credits.
Availability	Local, full, and lightweight clients.
Syntax	*maxValue* = *object*.**MaxDeficitCredits** **Element** — **Description** *maxValue* — The maximum number of deficit credits, which is always **0**. *creditObject* — The *CreditInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

UsedCredits (Deprecated)

Summary	Returns the number of devices defined, excluding NSDs and routers. **Note**: In previous LNS releases, this property indicted the the number of credits used.
Availability	Local, full, and lightweight clients.
Syntax	*maxValue* = *object*.**MaxDeficitCredits** **Element** — **Description** *usedValue* — The number of of devices defined, excluding NSDs and routers. *creditObject* — The *CreditInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DatabaseValidationReport

The *DatabaseValidationReport* object is returned after a database validation has been completed, and it contains information summarizing the results of the database validation. You can initiate a database validation on a network by calling the *Validate* method of the *Network* object.

- If the database passed the validation, no errors or inconsistencies were discovered during the procedure, or OpenLNS was able to repair all the errors it discovered. In this case, the *PassedValidation* property will be set to True.
- If the database does not pass the validation, then some errors still exist in the database after the validation. In this case, the *PassedValidation* property will be set to False.

All the other properties contain information describing the errors that were discovered in the database during the validation. This information includes how many errors were discovered during the validation, how many of the errors OpenLNS was able to repair, how many of the errors OpenLNS was not able to repair, and the *ErrorSummaries* property, which contains a collection of *DatabaseValidationErrorSummary* objects. Each *DatabaseValidationErrorSummary* object contains information describing a certain type of error that was discovered during the database validation.

The following table summarizes the *DatabaseValidationReport* object.

Description	A single point of data in a LONWORKS network.
Added to API	LNS Release 3.20.
Accessed Through	*Network* object.
Default Property	None.
Methods	• *Export*
Properties	• *ClassId* • *ErrorSummaries* • *Parent* • *PassedValidation* • *RepairedErrors* • *TotalObjectsValidated* • *UnrepairedErrors*

Methods

The *DatabaseValidationReport* object contains the following methods:

- *Export*

Export

Summary	Contains information summarizing the results of a database validation that was performed on a network. You can use this method to export the contents of the database validation report to a specified directory. The contents of the validation report will be exported in XML format. You must enter the directory and filename that will contain the report as a valid system path. If OpenLNS is unable to open the target directory for writing, or if the specified filename already exists and the *overwriteExisting* element is set to False, the **LCA, #149 lcaErrorInvalidExportFilename** exception will be thrown.
Availability	Local clients.
Syntax	*validationReport*.**Export** *path*, *overwriteExisting* **Element** — **Description** *validationReport* — The *DatabaseValidationReport* object to be acted upon object being acted upon. *path* — A string containing the path and

		filename to where the file is to be exported. This must be a valid system path
	overwriteExisting	This element is implemented if you specify a path and filename that already exists when you call this method. It has a Boolean value indicating whether the exported report should overwrite any existing files. **TRUE**. The existing file is overwritten. **FALSE**. The existing file is not overwritten, and an exception is thrown.
Added to API	LNS Release 3.20.	

Properties

The *DatabaseValidationReport* object contains the following properties:

- *ClassId*
- *ErrorSummaries*
- *Parent*
- *PassedValidation*
- *RepairedErrors*
- *TotalObjectsValidated*
- *UnrepairedErrors*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *DatabaseValidationReport* object in the *ConstClassIds* constant: **89 lcaClassIdDatabaseValidationReport** *object* — The object to be acted on.

Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ErrorSummaries

Summary	Contains the *DatabaseValidationErrorSummaries* collection object returned by the database validation report. The *DatabaseValidationErrorSummaries* collection object contains a group of *DatabaseValidationErrorSummary* objects. Each *DatabaseValidationSummary* object contains information about a certain type of error that was discovered during the database validation. You can use the information stored in these objects to troubleshoot the problems that may still exist in your database after a validation.
Availability	Local clients.
Syntax	*summariesCollection* = *validationReport.ErrorSummaries* **Element** — **Description** *summariesCollection* — The *DatabaseValidationErrorSummaries* collection object returned. *validationReport* — The *DatabaseValidationReport* being acted upon.
Data Type	*DatabaseValidationErrorSummaries* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.

Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PassedValidation

Summary	Indicates whether the network passed the database validation.
Availability	Local clients.
Syntax	*validated = validationReport.PassedValidation* **Element** — **Description** *validated* — A Boolean value indicating whether the network passed the validation procedure. **TRUE**. The network passed the database validation. This will occur if no errors were discovered during the network validation, or if all the errors discovered during the validation were repaired. **FALSE**. The network database has problems that were not repaired during the validation. *validationReport* — The *DatabaseValidationReport* object to be acted upon.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.20.

RepairedErrors

Summary	Contains the number of errors that were successfully repaired by OpenLNS during the database validation. You can initiate a database validation on a network by calling the *Validate* method on the *Network* object. To have LNS attempt to repair the errors it discovers during the validation, select **lcaDbValidateAndRepair (1)** as the *validationFlags* element when you call the *Validate* method. You can determine the number of errors that were not repaired during the validation by reading the *UnrepairedErrors* property.
Availability	Local clients.

Syntax	*repairedCount = validationReport.RepairedErrors* **Element** **Description** *repairedCount* The number of errors discovered during the database validation that were repaired. *validationReport* The *DatabaseValidationReport* object to be acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

TotalObjectsValidated

Summary	Contains the number of objects that were validated during the database validation. An object is validated if it can be accessed through its parent object in the database. Unvalidated objects are those that cannot be accessed through their parent object in the database. These are considered to be "orphan objects." A *DatabaseValidationErrorSummary* object will be included for the orphan objects discovered in the database.
Availability	Local clients.
Syntax	*validatedCount = validationReport. TotalObjectsValidated* **Element** **Description** *validatedCount* The number of objects that were validated during the database validation. *validationReport* The *DatabaseValidationReport* object to be acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

UnrepairedErrors

Summary	Contains the number of errors discovered during the database validation that were not repaired by OpenLNS. This includes all errors OpenLNS did not attempt to repair, and all errors OpenLNS was not able to repair. You can initiate a database validation on a network by calling the *Validate* method on the *Network* object. To have LNS attempt to repair the errors it discovers during the validation, select **lcaDbValidateAndRepair (1)** as the *validationFlags* element when you call the *Validate* method. You can determine the number of errors that were repaired during the validation by reading the *RepairedErrors* property.
Availability	Local clients.
Syntax	*repairedCount = validationReport.UnrepairedErrors* **Element** — **Description** *repairedCount* — The number of errors discovered during the database validation that were not repaired. *validationReport* — The *DatabaseValidationReport* object to be acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

DatabaseValidationErrorInstance

The *DatabaseValidationErrorInstance* object is returned by the *GetInstance* method of the *DatabaseValidationErrorSummary* object. It contains information indicating whether OpenLNS was able to repair a specific error during a database validation. The returned *DatabaseValidationErrorInstance* object also enables you to identify the object that caused the error, including the class, internal ID, and name of the object, as well as its parent object.

You can validate a database by calling the *Validate* method on the *Network* object. The database validation generates a *DatabaseValidationReport* that includes the *DatabaseValidationErrorSummaries* object, which contains a collection of *DatabaseValidationErrorSummary* objects. Each *DatabaseValidationErrorSummary* object contains information about a specific error type that was encountered during the database validation. This information includes includes the number of times the error type was encountered, whether OpenLNS can repair the error, and whether OpenLNS attempted to repair the error.

If the *ObjectDetailsAvailable* property of the *DatabaseValidationErrorSummary* object is set to True, you can use the *GetInstance* method to access specific details about each instance of the error type summarized by this object.

The following table summarizes the *DatabaseValidationErrorSummary* object.

Description	Indicates whether OpenLNS was able to repair a specific error during a database validation.
Added to API	LNS Release 3.20.
Accessed Through	*DatabaseValidationErrorSummaries* object.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *ErrorType* • *ObjectClassInvolved* • *ObjectInternalId* • *ObjectName* • *Parent* • *ParentObjectClass* • *ParentObjectInternalId* • *ParentObjectName* • *Repairable* • *RepairAttempted* • *Repaired*

Methods

The *DatabaseValidationErrorInstance* object does not contain any methods:

Properties

The *DatabaseValidationErrorInstance* object contains the following properties:

- *ClassId*
- *ErrorType*
- *ObjectClassInvolved*
- *ObjectInternalId*
- *ObjectName*
- *Parent*
- *ParentObjectClass*
- *ParentObjectInternalId*
- *ParentObjectName*
- *Repairable*
- *RepairAttempted*
- *Repaired*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *DatabaseValidationErrorInstance* object in the *ConstClassIds* constant: **92 lcaClassIdDatabaseValidationErrorInstance** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ErrorType

Summary	Indicates the type of error refernced by this *DatabaseValidationErrorInstance* object. The other properties of the *DatabaseValidationErrorInstance* and *DatabaseValidationErrorSummary* objects contain additional information that may be useful when troubleshooting these errors. See these objects for more information.
Availability	Local clients.
Syntax	*errorType* = *validationObject.ErrorType* **Element** — **Description** *errorType* — The error type referenced by the *DatabaseValidationErrorInstance* object. The values that may be returned as this element, which are contained in the *ConstDbValidationErrorType* constant, are as follows: **0 lcaDbValidationErrorOrphanDevice** An orphan *AppDevice* object was found in the database. An orphan object is one that is no longer accessible through its parent object. **1 lcaDbValidationErrorOrphanObject** An orphan object was found in the database. An orphan object is one that is no longer accessible through its parent object. **2 lcaDbValidationErrorDuplicateDynamicNV**

		A duplicate dynamic network variable was found on a device's interface. **3 lcaDbValidationError MissingNV** An invalid reference to a network variable, or a missing network variable, was discovered on a dynamic interface. **4 lcaDbValidationError MissingLMO** An invalid reference to a *LonMark* object, or a missing LonMark object, was discovered on a dynamic interface. **5 lcaDbValidationError MissingCP** An invalid reference to a configuration property, or a missing configuration property, was discovered. **6 lcaDbValidationError MissingMT** An invalid reference to a message tag, or a missing message tag, was discovered. **7 lcaDbValidationErrorBad DbRecord** A corrupted or invalid record was discovered in the global database. **8 lcaDbValidationErrorBadLink** An invalid reference to another object in the database was discovered. **9 lcaDbValidationError BadSystemPointer** An invalid reference to the *System* object from another object was discovered in the database. **10 lcaDbValidationError BadBaseObjectData** Invalid base-object data was discovered. This is an internal error. **11 lcaDbValidationErrorBad ObjectData** Invalid object data was discovered. This is an internal error. **12 lcaDbValidationErrorDuplicateDynamicMT** Duplicate dynamic message tag data was

		discovered. This is an internal error. **13 lcaDbValidationErrorDuplicateDynamicLMO** Duplicate LonMark object data was discovered. This is an internal error. **14 lcaDbValidationErrorBadInterfaceCount** Incorrect interface count data was discovered. This is an internal error. **15 lcaDbValidationErrorDuplicate StaticNV** Duplicate static network variable data was discovered. This is an internal error. **16 lcaDbValidationErrorMissing Subsystem** An invalid reference to a subsystem, or a missing subsystem, was discovered. This is an internal error. **17 lcaDbValidationErrorMissing AppDevice** An invalid reference to a device, or a missing device, was discovered. This is an internal error. **18 lcaDbValidationErrorMissing Router** An invalid reference to a router, or a missing router, was discovered. **19 lcaDbValidationError DuplicateStaticLMO** Duplicate static functional block was discovered. **20 lcaDbValidationError DuplicateObject** Duplicate object data was discovered. **128 lcaDbValidationErrorNss RecordIntegrity** This is an internal error. **129 lcaDbValidationError NssUniqueKeyIntegrity** This is an internal error. **130 lcaDbValidationErrorNss NonUniqueKeyIntegrity** This is an internal error. **131 lcaDbValidationErrorNssSet**

		Integrity This is an internal error. **132 lcaDbValidationErrorNssLink Integrity** This is an internal error.
	validationObject	The *DatabaseValidationErrorInstance* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

ObjectClassInvolved

Summary	Indicates the class ID of the object that caused the error, which is referenced by this *DatabaseValidationErrorInstance* object. For example, if the error type is **lcaDbValidationErrorOrphanObject**, this property indicates the class ID of the orphan object that was discovered. The values returned by this property map to the values of the *ConstClassIds* constant; therefore you can use the *ConstClassIds* constant to determine the object type the class ID applies to. For example, if this property returns **7**, the object is an *AppDevice* object. See the *ComponentClassID* property of the *ComponentApp* object for a list of the class IDs. You can determine the name of the object that caused the error by reading the *ObjectName* property.	
Availability	Local clients.	
Syntax	*classID = errorInstance.ObjectClassInvolved* **Element** / **Description** *classID* — The class ID of the object that caused the error. *errorInstance* — The *DatabaseValidationErrorInstance* object being acted upon.	
Data Type	Integer.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

ObjectInternalId

Summary	Indicates the internal ID of the object that caused the error, which is referenced by this *DatabaseValidationErrorInstance* object. For example, if the error type is lcaDbValidation**ErrorMissingLMO**, this property would contain the internal ID of the missing LonMarkObject.
Availability	Local clients.
Syntax	*internalId = errorInstance.ObjectInternalId* **Element** — **Description** *internalId* — The internal ID of the object that caused the error. *errorInstance* — The *DatabaseValidationErrorInstance* object being acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

ObjectName

Summary	Indicates the name of the object that caused the error, which is referenced by this *DatabaseValidationErrorInstance* object. For example, if the error type is **lcaDbValidationErrorMissingNV**, this property would contain the name of the missing *NetworkVariable* object. You can determine the type of error referenced by this object by reading the *ErrorType* property.
Availability	Local clients.
Syntax	*name = errorInstance.ObjectName* **Element** — **Description** *name* — The name of the object that caused the error. *errorInstance* — The *DatabaseValidationErrorInstance* object being acted upon.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* **=** *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ParentObjectClass

Summary	Indicates the class ID of the parent of the object that caused the error, which is referenced by this *DatabaseValidationErrorInstance* object. For example, if the error type is **lcaDbValidationErrorMissingLMO**, this property would contain the class ID of the parent of the missing LonMarkObject (the object the missing LonMarkObject should be accessed through). The values returned by this property map to the values of the *ConstClassIds* constant; therefore you can use the *ConstClassIds* constant to determine the object type the class ID applies to. For example, if this property returns **9**, the object is a *Router* object. See the *ComponentClassID* property of the *ComponentApp* object for a list of the class IDs. You can determine the type of error referenced by the *DatabaseValidationErrorInstance* object by reading the *ErrorType* property. You can determine the name of the parent of the object that caused the error by reading the *ParentObjectName* property. If the object associated with the error does not have a parent object (the error type is **lcaDbValidationErrorOrphanObject** or **lcaDbValidationErrorOrphanDevice**), this property will contain an empty string.
Availability	Local clients.
Syntax	*parentClass = errorInstance.ParentObjectClass* **Element** — **Description** *parentClass* — The class ID of the parent of the object that caused the error. *errorInstance* — The *DatabaseValidationErrorInstance* object being acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

ParentObjectInternalId

Summary	Indicates the internal ID of the parent of the object that caused the error, which is referenced by this *DatabaseValidationErrorInstance* object. For example, if the error type is **lcaDbValidationErrorMissingNv**, this property would contain the internal ID of the parent object of the missing *NetworkVariable* object (the name of the object the missing *NetworkVariable* object should be accessed through). You can determine the type of error referenced by the *DatabaseValidationErrorInstance* object by reading the *ErrorType* property. You can determine the name of the parent of the object that caused the error by reading the *ParentObjectName* property. If the object associated with the error does not have a parent object (the error type is **lcaDbValidationErrorOrphanObject** or **lcaDbValidationErrorOrphanDevice**), this property will contain an empty string.
Availability	Local clients.
Syntax	*parentInternalId* = *errorInstance*.**ParentObjectInternalId** **Element** — **Description** *parentInternalId* — The internal ID of the parent of the object that caused the error. *errorInstance* — The *DatabaseValidationErrorInstance* object being acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

ParentObjectName

Summary	Indicates the name of the parent of the object that caused the error, which is referenced by this *DatabaseValidationErrorInstance* object. For example, if the error type is **lcaDbValidationErrorMissingNv**, this property would contain the name of the parent object of the missing *NetworkVariable* object (the name of the object the missing *NetworkVariable* object should be accessed through). You can determine the type of error referenced by the *DatabaseValidationErrorInstance* object by reading the *ErrorType* property. You can determine the name of the parent of the object that caused the error by reading the *ParentObjectName* property. If the object associated with the error does not have a parent object (the error type is **lcaDbValidationErrorOrphanObject** or **lcaDbValidationErrorOrphanDevice**), this property will contain an empty string.
Availability	Local clients.
Syntax	*parentName = errorInstance.ParentObjectName* **Element** — **Description** *parentName* — The name of the parent of the object that caused the error. *errorInstance* — The *DatabaseValidationErrorInstance* object being acted upon.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Repairable

Summary	Indicates whether OpenLNS can repair errors of this type as part of the database validation procedure.
Availability	Local clients.

<table>
<tr><td>Syntax</td><td>isRepairable = validationObject.Repairable

Element — Description

validationObject — The DatabaseValidationErrorInstance object being acted upon.

isRepairable — A Boolean value indicating whether OpenLNS can repair errors of this type.

TRUE. OpenLNS can automatically repair this type of error (or this specific instance of an error type) as part of the database validation procedure.

To have LNS repair the error, call the Validate method to initiate the database validation and set the validationFlags element to lcaDbValidateAnd Repair (1).

This will occur if no errors were discovered during the network validation, or if all the errors discovered during the validation were repaired.

FALSE. OpenLNS can not automatically repair this type of error.</td></tr>
<tr><td>Data Type</td><td>Boolean.</td></tr>
<tr><td>Read/Write</td><td>Read only.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.20.</td></tr>
</table>

RepairAttempted

Summary	Indicates whether OpenLNS attempted to repair errors of this type during the validation. If you call the *Validate* method to initiate a database validation procedure and set the *validationFlags* element to **lcaDbValidateAndRepair (1)**, OpenLNS will attempt to repair some errors it finds automatically. LNS may not be able to repair all error types or all the specific instances of an error type it encounters during the validation. You can use this property to determine whether OpenLNS attempted to repair a specific error or a specific error type. You can determine if OpenLNS was able to repair a specific instance of an error by reading the *Repaired* property of the error's DatabaseSummaryErrorInstance object.
Availability	Local clients.
Syntax	*repairAttempted = errorReport.RepairAttempted* **Element** — **Description** *errorReport* — The *DatabaseValidationErrorInstance* object being acted upon. *repairAttempted* — A Boolean value indicating whether OpenLNS attempted to repair errors of this type during the validation. **TRUE**. OpenLNS attempted to repair errors of this type during the validation. **FALSE**. OpenLNS did not attempt to repair errors of this type during the validation.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Repaired

Summary	Indicates whether LNS repaired the error referenced by this *DatabaseValidationErrorInstance* object, and the action taken by OpenLNS to make the repair.
Availability	Local clients.

<table>
<tr><td>Syntax</td><td>repaired = errorReport.Repairable
Element — Description
errorReport — The DatabaseValidationErrorInstance object being acted upon.
repaired — This element indicates the action LNS performed to repair the error.
The possible values that can be returned as this element, which are contained in the ConstDbValidationRepairLevel constant, are as follows:
0 lcaDbValidationNotRepaired
No repair was performed on the object that caused the error, or the repair was not completed successfully.
1 lcaDbValidationObjectDeleted
The object causing the error was deleted. This may be the case if there is an orphan object in the database, or if there is an invalid object reference in the database.
2 lcaDbValidationObjectSetToDefaults
The object causing the error was deleted, and then re-created with default values. This may be the case if the error was caused by a NetworkVariable, ConfigProperty, or LonMarkObject object.
3 lcaDbValidationObjectFullyRepaired
The object causing the error was completely repaired, with no loss of data. This may be the case if the error was caused by a bad link between two objects in the database.</td></tr>
<tr><td>Data Type</td><td>Long.</td></tr>
<tr><td>Read/Write</td><td>Read only.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.20.</td></tr>
</table>

DatabaseValidationErrorSummaries

The *DatabaseValidationErrorSummaries* property contains a collection of *DatabaseValidationErrorSummary* objects. To retrieve a *DatabaseValidationErrorSummary* object from this collection, use the *Item* property and specify the object to be retrieved by its

index number. The index number will be in the range of 1 to *x*, where *x* represents the value stored in the *Count* property.

The following table summarizes the *DatabaseValidationErrorSummaries* object.

Description	A collection of *DatabaseValidationErrorSummary* objects.
Added to API	LNS Release 3.0.
Accessed Through	*DatabaseValidationErrorSummary* object.
Default Property	*Item*.
Methods	None.
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *DatabaseValidationErrorSummaries* object does not contain any methods.

Properties

The *DatabaseValidationErrorSummaries* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *DatabaseValidationErrorSummaries* object in the *ConstClassIds* constant: **91 lcaClassIdDatabaseValidationErrorSummaries** *object* — The object to be acted on.

Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an object from a collection. You can retrieve an object from its collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The object retrieved from the collection. *collObject* — The collection object to be acted on. *index* — A Long type specifying the ordinal index of the object to retrieve. *stringExpression* — A string type specifying the name of the object to retrieve.
Data Type	Object.

Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.

Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

DatabaseValidationErrorSummary

The *DatabaseValidationErrorSummary* object contains information describing a certain type of error that was encountered during the database validation. This includes the number of times the error type was encountered in the database, whether OpenLNS can repair errors of that type, and whether OpenLNS attempted to repair any errors of that type.

If the *ObjectDetailsAvailable* property is set to True, you can use the *GetInstance* method to access specific details about each instance of the error type summarized by this object

The following table summarizes the *DatabaseValidationErrorSummary* object.

Description	A single point of data in a LONWORKS network.
Added to API	LNS Release 3.20.
Accessed Through	*DatabaseValidationErrorSummaries* object.
Default Property	None.
Methods	• *GetInstance*
Properties	• *ClassId* • *ErrorType* • *NumberOfInstances* • *ObjectDetailsAvailable* • *Parent* • *Repairable* • *RepairAttempted*

Methods

The *DatabaseValidationErrorSummary* object contains the following methods:

- *GetInstance*

GetInstance

Summary	Retrieves specific information about each instance of an error type. For some types of errors, you will not be able to access specific information about each instance of the error type. You can check if specific information about each instance is by reading the

	ObjectDetailsAvailable property. If you specify an invalid number as the *index* element when invoking this method, the **LCA, #15 lcaErrInvalidCollectionIndex** exception will be thrown.
Availability	Local clients.
Syntax	*errorInstance* = *validationSummary.GetInstance index* **Element** — **Description** *errorInstance* — The *DatabaseValidationErrorInstance* object returned by the method. This object will contain detailed information about an error discovered during the validation. *validationSummary* — The *DatabaseValidationErrorSummary* object being acted upon. *index* — The index value assigned to the error to be accessed. Each error is assigned an index value in the range of 1 to *x*, where *x* represents the value assigned to the *NumberOfInstances* property of the *DatabaseValidationErrorSummary* object.
Added to API	LNS Release 3.20.

Properties

The *DatabaseValidationErrorSummary* object contains the following properties:

- *ClassId*
- *ErrorType*
- *NumberOfInstances*
- *ObjectDetailsAvailable*
- *Parent*
- *Repairable*
- *RepairAttempted*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId**

	Element	**Description**
	classIdValue	The object class of the object. The following value is defined for the *DatabaseValidationErrorSummary* object in the *ConstClassIds* constant: **91 lcaClassId DatabaseValidationErrorSummary**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

ErrorType

Summary	Indicates the type of error a *DatabaseValidationErrorSummary* object represents. The other properties of the *DatabaseValidationErrorInstance* and *DatabaseValidationErrorSummary* objects contain additional information that may be useful when troubleshooting these errors. See these objects for more information.	
Availability	Local clients.	
Syntax	*errorType = validationObject.ErrorType*	
	Element	**Description**
	errorType	The error type that the *DatabaseValidationErrorSummary* object represents. The values that may be returned as this element, which are contained in the *ConstDbValidationErrorType* constant, are as follows: **0 lcaDbValidationErrorOrphan Device** An orphan *AppDevice* object was found in the database. An orphan object is one that is no longer accessible through its parent object. **1 lcaDbValidationErrorOrphan Object** An orphan object was found in the database. An orphan object is one that is no longer accessible through its parent object.

		2 lcaDbValidationError DuplicateDynamicNV A duplicate dynamic network variable was found on a device's interface. **3 lcaDbValidationError MissingNV** An invalid reference to a network variable, or a missing network variable, was discovered on a dynamic interface. **4 lcaDbValidationError MissingLMO** An invalid reference to a *LonMark* object, or a missing LonMark object, was discovered on a dynamic interface. **5 lcaDbValidationError MissingCP** An invalid reference to a configuration property, or a missing configuration property, was discovered. **6 lcaDbValidationError MissingMT** An invalid reference to a message tag, or a missing message tag, was discovered. **7 lcaDbValidationErrorBad DbRecord** A corrupted or invalid record was discovered in the global database. **8 lcaDbValidationErrorBadLink** An invalid reference to another object in the database was discovered. **9 lcaDbValidationError BadSystemPointer** An invalid reference to the *System* object from another object was discovered in the database. **10 lcaDbValidationError BadBaseObjectData** Invalid base-object data was discovered. This is an internal error. **11 lcaDbValidationErrorBad ObjectData** Invalid object data was discovered. This is an internal error.

		12 lcaDbValidationErrorDuplicateDynamicMT Duplicate dynamic message tag data was discovered. This is an internal error. **13 lcaDbValidationErrorDuplicateDynamicLMO** Duplicate LonMark object data was discovered. This is an internal error. **14 lcaDbValidationErrorBadInterfaceCount** Incorrect interface count data was discovered. This is an internal error. **15 lcaDbValidationErrorDuplicateStaticNV** Duplicate static network variable data was discovered. This is an internal error. **16 lcaDbValidationErrorMissingSubsystem** An invalid reference to a subsystem, or a missing subsystem, was discovered. This is an internal error. **17 lcaDbValidationErrorMissingAppDevice** An invalid reference to a device, or a missing device, was discovered. This is an internal error. **18 lcaDbValidationErrorMissingRouter** An invalid reference to a router, or a missing router, was discovered. **19 lcaDbValidationErrorDuplicateStaticLMO** Duplicate static functional block was discovered. **20 lcaDbValidationErrorDuplicateObject** Duplicate object data was discovered. **128 lcaDbValidationErrorNssRecordIntegrity**

		This is an internal error. **129 lcaDbValidationError NssUniqueKeyIntegrity** This is an internal error. **130 lcaDbValidationErrorNss NonUniqueKeyIntegrity** This is an internal error. **131 lcaDbValidationErrorNssSet Integrity** This is an internal error. **132 lcaDbValidationErrorNssLink Integrity** This is an internal error.
	validationObject	The *DatabaseValidationErrorSummary* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

NumberOfInstances

Summary	Indicates the number of times this type of error was encountered in the database. To determine the type of error represented by this object, read the *ErrorType* property of the *DatabaseValidationErrorSummary* object	
Availability	Local clients.	
Syntax	*errorCount = validationObject.NumberOfInstances*	
	Element	**Description**
	errorCount	The number of errors of this type that were discovered during the database validation.
	validationObject	The *DatabaseValidationErrorSummary* object being acted upon.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

ObjectDetailsAvailable

Summary	Indicates whether you can retrieve specific information about each instance of the error type from the object. The availability of this information depends primarily on the error type that this *DatabaseValidationErrorSummary* object represents. To determine the error type, read the *ErrorType* property of the *DatabaseValidationErrorSummary* object.
Availability	Local clients.
Syntax	*detailsAvailable* = *validationSummary*.*ObjectDetailsAvailable* **Element** — **Description** *detailsAvailable* — A Boolean value that indicates whether you can retrieve specific information about each instance of the error type from the object. **TRUE**. You can obtain detailed information about each instance of this error type from the object. **FALSE**. You can only obtain summary information for this error type. *validationSummary* — The *DatabaseValidationErrorSummary* object being acted upon.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.

Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Repairable

Summary	Indicates whether OpenLNS can repair errors of this type as part of the database validation procedure.
Availability	Local clients.
Syntax	*isRepairable = validationObject.Repairable* **Element** — **Description** *validationObject* — The *DatabaseValidationErrorSummary* object being acted upon. *isRepairable* — A Boolean value indicating whether OpenLNS can repair errors of this type. **TRUE**. OpenLNS can automatically repair this type of error (or this specific instance of an error type) as part of the database validation procedure. To have LNS repair the error, call the *Validate* method to initiate the database validation and set the *validationFlags* element to **lcaDbValidateAndRepair (1)**. This will occur if no errors were discovered during the network validation, or if all the errors discovered during the validation were repaired. **FALSE**. OpenLNS can not automatically repair this type of error.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.20.

RepairAttempted

Summary	Indicates whether OpenLNS attempted to repair errors of this type during the validation. If you call the *Validate* method to initiate a database validation procedure and set the *validationFlags* element to **lcaDbValidateAndRepair (1)**, OpenLNS will attempt to repair some errors it finds automatically. LNS may not be able to repair all error types or all the specific instances of an error type it encounters during the validation. You can use this property to determine whether OpenLNS attempted to repair a specific error or a specific error type. You can determine if OpenLNS was able to repair a specific instance of an error by reading the *Repaired* property of the error's *DatabaseSummaryErrorInstance* object.
Availability	Local clients.
Syntax	*repairAttempted = errorReport.RepairAttempted* **Element** — **Description** *errorReport* — The *DatabaseValidationErrorSummary* object being acted upon. *repairAttempted* — A Boolean value indicating whether OpenLNS attempted to repair errors of this type during the validation. **TRUE**. OpenLNS attempted to repair errors of this type during the validation. **FALSE**. OpenLNS did not attempt to repair errors of this type during the validation.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.20.

DataPoint

A *DataPoint* object represents a single point of data in a LONWORKS network. You can obtain a data point through any *MsgMonitorPoint*, *NvMonitorPoint*, *NetworkVariable*, or *ConfigProperty* object. Once acquired, each data point is implicitly bound to the object from which it was obtained. This object is referred to as the data point's *source object*.

You can then use the data point to read and write to the value of the source object. Each data point has three properties you can use to read and write to the data point's value: the *FormattedValue* property, the *RawValue* property, and the *Value* property. Each of these properties represents the same value, but each one is formatted differently.

When you read and write to any of these properties, OpenLNS will also read or write the data point's source object (i.e. the value of the source object in the OpenLNS database and on

the network device containing the source object) by default. You can change this behavior by setting the *AutoRead* and *AutoWrite* properties of the *DataPoint* object to False. If these properties are set to False, OpenLNS will not update the source object every time the value properties of the data point are updated. However, in this case, you can still use data point's *Write* method to write the value stored in these properties to the data point's source object. You can also use the *Read* method to update the value stored in these properties with the current value of the source object.

Formatting of each data point is handled locally. As a result, formatting changes made to the value of a data point's source object by your application do not affect other clients that are attempting to read the value of the same network variable, monitor point or configuration property. This will eliminate any confusion that may be caused by formatting changes made by another client application.

The following table summarizes the *DataPoint* object.

Description	A single point of data in a LONWORKS network.
Added to API	LNS Release 3.0.
Accessed Through	*ConfigProperty* *MsgMonitorPoint* *NvMonitorPoint* *NetworkVariable*
Default Property	None.
Methods	• *GetField* • *Read* • *Write*
Properties	• *AutoRead* • *AutoWrite* • *ClassId* • *FieldCount* • *FieldName* • *FormatSpec* • *FormattedValue* • *MaxValue* • *MessageCode* • *MinValue* • *Parent* • *SourceIndex* • *SourceOptions* • *TypeName* • *Value*

Methods

The *ConfigProperties* object contains the following methods:

- *GetField*
- *Read*

- *Write*

GetField

Summary	Gets a *DataPoint* object containing one field if the *DataPoint* object contains a structure. The *AutoWrite* property of the returned *DataPoint* is set to False; therefore, the *Write* method must be called explicitly to write the information to the network variable. The number of fields in a *DataPoint* object can be determined from the *FieldCount* property. The name of a field can be determined by that field's *FieldName* property.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*dpField* = *dpObject*.**GetField**(*indexName*) **Element** — **Description** *dpField* — The *DataPoint* object returned. This represents the data point field requested in the call to *GetField*. *dpObject* — The *DataPoint* object to be acted on. *indexName* — The index number or name of the data point field to be returned.
Added to API	LNS Release 3.20.

Read

Summary	Updates the *Value*, *RawValue*, and *FormattedValue* properties with the current value of the data point's source object. You can use either of these properties to read and write to the data point's value; however, they are formatted differently. See these individual properties for more information. After the *Read* method is called, OpenLNS will update the value of these properties to match the value of the data point's source object. For example, if the data point was obtained through a network variable, the network variable will be fetched from the network, and the three properties would be updated to match the value reported by the physical device on the network. This will ensure that the data point returns a current value for its source object. If the *AutoRead* property is set to True, OpenLNS will call the Read method automatically each time you read the *RawValue*, *FormattedValue*, or *Value* properties.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*dpObject*.**Read** **Element** **Description** *dpObject* The *DataPoint* object being acted upon.
Added to API	LNS Release 3. 0.

Write

Summary	Updates the value of the source object controlled by this data point to match the current value stored in the data point's *Value*, *RawValue*, and *FormattedValue* properties. Writing to either of the *Value*, *RawValue*, and *FormattedValue* properties updates the values of all three so that they always match. However, you must call the *Write* method to propagate these new values to the source object. After the *Write* method is called, OpenLNS will update the value of the data point's source object to match the value of these properties. For example, if the data point was obtained through a network variable, the value of the network variable on the physical device on the network would be updated to match the values of these properties after the *Write* method was called. This ensures that the data point returns a current value for its source object. If the *AutoWrite* property is set to True, this method will automatically be called every time the *RawValue*, *FormattedValue*, or *Value* properties are written to. **Note**: If you call this method on a *DataPoint* obtained from a *NetworkVariable* or *NvMonitorPoint* object before reading its value from the network (either explicitly by calling the *Read* method, or implicitly when the *AutoRead* property is set to True), or before setting the Data Point's value with the *RawValue*, *FormattedValue* or *Value* properties, the **DS, #370 lcaErrLnsDsInvalidArg** exception will be thrown. This indicates that the *DataPoint*'s buffer is empty.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*dpObject.Write* **Element** **Description** *dpObject* The *DataPoint* object being acted upon.
Added to API	LNS Release 3.20.

Properties

The *DataPoint* object contains the following properties:

- *AutoRead*

- *AutoWrite*
- *ClassId*
- *FieldCount*
- *FieldName*
- *FormatSpec*
- *FormattedValue*
- *MaxValue*
- *MessageCode*
- *MinValue*
- *Parent*
- *SourceIndex*
- *SourceOptions*
- *TypeName*
- *Value*

AutoRead

Summary	Determines whether the *Read* method will be called automatically whenever the *Value*, *FormattedValue*, or *RawValue* properties are read. You can use either of these properties to read the data point's value; however, they are formatted differently. See these individual properties for more information.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*autoRead = dpObject.AutoRead* **Element** — **Description** *dpObject* — The *DataPoint* object being acted upon. *autoRead* — A Boolean value. • True. The Read method is called automatically each time you read the data point's *Value*, *FormattedValue*, or *RawValue* properties. This ensures that the value returned by the data point, through any of the three value properties, always represents the current value of the source object. • False. You must call the *Read* method manually. The default is True. The only exception is when the *DataPoint* object is to be used for a response (see the *OnMsgMonitorPointUpdateEvent* event), or when the *DataPoint* object represents a field (see the *GetField*

	method).
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

AutoWrite

Summary	Determines whether the *Write* method will be called automatically whenever the *FormattedValue*, *RawValue*, or *Value* properties are written. You can use either of these properties to write to the data point's value; however, they are formatted differently. See these individual properties for more information.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*autoWrite = dpObject.AutoWrite* **Element** — **Description** *dpObject* — The *DataPoint* object being acted upon. *autoWrite* — A Boolean value. • True. The *Write* method is called automatically each time you write to the data point's *Value*, *FormattedValue*, or *RawValue* properties. This ensures that the value returned by the data point, through any of the three value properties, always represents the current value of the source object. • False. You must call the *Write* method manually to ensure the values in all three properties remain consistent. This property defaults to True. The only exception is when the *DataPoint* object represents a field (see the *GetField* method for more information).
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *DataPoint* object in the *ConstClassIds* constant: **72 lcaClassIdDataPoint** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

FieldCount

Summary	Indicates the number of fields contained by this data point. If a *DataPoint* object contains multiple fields, each field is represented by its own *DataPoint* object. You can use the *DataPoint* object's *GetField* method to get one of the *DataPoint* object's fields. The fields can be indexed by number and by name. The *FieldCount* property contains the name of the field, if the *DataPoint* object represents one.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*fieldCountValue* = *dpObject*.**FieldCount** **Element** — **Description** *fieldCountValue* — The field count. *dpObject* — The *DataPoint* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

FieldName

Summary	Indicates the name of the field if the *DataPoint* object contains multiple fields. If a *DataPoint* object contains multiple fields, each field is represented by its own *DataPoint* object. Each *DataPoint* object representing a field has a name; for example, a *DataPoint* which contained the time could have three fields called "hour", "minute", and "second". When the *GetField* method is called, the fields can be accessed either by name or by index number. The *FieldCount* property indicates how many fields a *DataPoint* object contains. **Note**: If this *DataPoint* object does not represent a field, this property will return an empty string.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*fieldName = dpObject.FieldName* **Element** — **Description** *fieldName* — The name of the field. *dpObject* — The *DataPoint* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

FormatSpec

Summary	Contains the format specification information for data read from the *FormattedValue* property in the *DataPoint* object.
	
	This property contains a *FormatSpec* object that determines the base type to use when reading the formatted values of the data point or monitor point. For data points acquired through NetworkVariable objects, the default settings for the *FormatSpec* object are determined based on the network variable's *TypeSpec* property.
	
	You can also use the *CurrentFormatLocale* property to determine what options OpenLNS will use when displaying the data stored in the *FormattedValue* property.
	
	The *FormatSpec* object contained within this property is not passed by reference. If you modify the values assigned to the properties of a local *FormatSpec* object, you must then explicitly assign the modified *FormatSpec* object back to the *FormatSpec* property of the *DataPoint* for the changes to take effect. This following code sample demonstrates this procedure:
	
	`Set fsObject = dpObject.FormatSpec`
	`fsObject.FormatName = "SNVT_temp_f#SI"`
	`Set dpObject.FormatSpec = fsObject`

Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*fsObject= dpObject.***FormatSpec** **Element** — **Description** *fsObject* — The *FormatSpec* object containing the format information. *dpObject* — The *DataPoint* object to be acted on.
Data Type	*FormatSpec* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

FormattedValue

Summary	Contains the value of the data point as a Variant. To read the value of a data point's source object (the value of the source object in the OpenLNS database and on the network device containing the source object) as a formatted value, call the *DataPoint* object's *Read* method, and then read this property. If the *AutoRead* is set to True, the *Read* method will be called automatically each time this property is read. To write the value stored in this property to a data point's source object (the value of the source object in the OpenLNS database and on the network device containing the source object), set this property to the desired value, and then call the *DataPoint* object's *Write* method. If the *AutoWrite* property is set to True, the *Write* method will automatically be called each time this property is written to. This property shares memory space with the *Value* and *RawValue* properties, which means that the information contained in all three properties represents the same value. The only difference between the three properties is that each them is formatted differently; therefore, when you write to the *FormattedValue* property, OpenLNS will update the values of the *Value* and *RawValue* properties automatically, to match the updated *FormattedValue* property. If the data point's source object is a network variable, you should note that reading the *Value* property always causes a poll and returns the resulting response, even if the network variable is bound to the host or if the Data Server is already polling. If the *Value* property is read, and the poll fails, an exception will be thrown. If the data point's source object is a configuration property, you can use the *ValueStatus* property to determine if the value stored in the OpenLNS database for the configuration property matches the value stored on the physical device. In addition, the setting of the *DataPoint* object's *SourceOptions* property affects how LNS accesses the value of the configuration property (either from the physical device containing the configuration property, or from the OpenLNS database).You should also be aware that some configuration properties contain a constant attribute. If a configuration property has the constant attribute set, it marks the configuration property as read-only. If this attribute is set and you attempt to write to the *FormattedValue* property, an exception will be thrown. However, in some cases, you can use the *ConstantAttribute* property to set or turn off this attribute using OpenLNS. You can use the *MinValue* and *MaxValue* properties to set the minimum and maximum possible values for the data point. If you write a value to the *FormattedValue* property that exceeds the range established by these properties, the

<table>
<tr><td></td><td>Formatter#16 lcaErrFormatIllegalDataOnUnformat exception will be thrown. You should note that OpenLNS only performs this range checking on scalar types. If you are using data points to read and write the value of a network variable or configuration property whose type is a structure and you want LNS to perform this range checking, you will need to create a separate data point for each field. You can create data points to represent each field in a given structure with the GetField method.

You can set options that determine how the information contained in the FormattedValue property will be formatted and displayed by writing to the FormatLocale object currently being used by the application. This includes options such as the format used to display dates and times and the measurement units used to display the value. You can determine which FormatLocale object is currently being used with the CurrentFormatLocale property of the ObjectServer object.</td></tr>
<tr><td>Availability</td><td>Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.</td></tr>
<tr><td>Syntax</td><td>value = dpObject.FormattedValue

Element — Description
value — The value of the data point as a Variant.
dpObject — The DataPoint object to be acted on.</td></tr>
<tr><td>Data Type</td><td>Variant.</td></tr>
<tr><td>Read/Write</td><td>Read/write.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.0.</td></tr>
</table>

MaxValue

Summary	Contains the maximum value that can be applied to the *FormattedValue* property of the *DataPoint* object. • If the data point's source object is a network variable, the default value for this property is determined based on the network variable's type, meaning that the default value will be the maximum value supported by the type. This value may be overridden by the network variable's entry in the Functional Profile Template used by the device hosting the network variable. • If the data point's source object is a network variable monitor point, the default value for this property is determined based on the source object's type (i.e. the type of the network variable being monitored). Note that the override value from the network variable's entry in the Functional Profile Template used by the device hosting the network variable is not available if the data point's source object is a network variable monitor point. • If the data point's source object is a configuration property, the default value for this property is taken from the type definitions of the configuration property type used by the source configuration property, from the configuration property's entry in the Functional Profile Template used by the device hosting the configuration property, or from the device's configuration property definition file. If OpenLNS is unable to retrieve a default maximum value for the data point, this property will be set to **lcaRangeValueNotSet**. **Note**: OpenLNS will only enforce the range established by this property when you write to the *FormattedValue* property of a data point. OpenLNS will not enforce this range when you write to the *Value* and *RawValue* properties of a data point. As a result, you should program your application to enforce this range for the *Value* and *RawValue* properties, and therefore avoid writing invalid data to the data point. In addition, the data stored in the *MaxValue* property is not unit-converted as is the data stored in the *FormattedValue* property. Instead, the data stored in the *MaxValue* property is scaled data that has not been unit-converted (similar to the data stored in the *Value* property). As a result, you need to consider the unit conversions applied by the OpenLNS Object Server when you write to the *FormattedValue* property to avoid violating the range established for the data point. You can determine what unit conversions are applied to a particular data point by reading the *UnitsMultiplier* and *UnitsAdder* properties of the *FormatSpec* object the data point is using.

Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*maxValue* = *dataPoint.SourceIndex* **Element** — **Description** *maxValue* — The maximum value that can be applied to the *FormattedValue* property. *dataPoint* — The *DataPoint* object being acted upon.
Data Type	Double.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

MessageCode

Summary	Contains the message code of the message associated with this data point. This property is only meaningful if the *DataPoint* object represents a message. This value has no effect if the data source is a configuration property or network variable; however, you can still read and write to the value. • If this *DataPoint* represents an input value, the message code indicates what message code appeared in the received message. • If this data point represents an output message, this is the code that will be sent with the message. • If this *DataPoint* represents a response to an incoming request, this property represents the response code of the response message.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*messageCode* = *dpObject*.**MessageCode** **Element** — **Description** *messageCode* — The message code of the message *dpObject* — The *DataPoint* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

MinValue

Summary	Contains the minimum value that can be applied to the *FormattedValue* property of the *DataPoint* object. • If the data point's source object is a network variable, the default value for this property is determined based on the network variable's type, meaning that the default value will be the maximum value supported by the type. This value may be overridden by the network variable's entry in the Functional Profile Template used by the device hosting the network variable. • If the data point's source object is a network variable monitor point, the default value for this property is determined based on the source object's type (i.e. the type of the network variable being monitored). Note that the override value from the network variable's entry in the Functional Profile Template used by the device hosting the network variable is not available if the data point's source object is a network variable monitor point. • If the data point's source object is a configuration property, the default value for this property is taken from the type definitions of the configuration property type used by the source configuration property, from the configuration property's entry in the Functional Profile Template used by the device hosting the configuration property, or from the device's configuration property definition file. If OpenLNS is unable to retrieve a default maximum value for the data point, this property will be set to **lcaRangeValueNotSet**. **Note**: OpenLNS will only enforce the range established by this property when you write to the *FormattedValue* property of a data point. OpenLNS will not enforce this range when you write to the *Value* and *RawValue* properties of a data point. As a result, you should program your application to enforce this range for the *Value* and *RawValue* properties, and therefore avoid writing invalid data to the data point. In addition, the data stored in the *MinValue* property is not unit-converted as is the data stored in the *FormattedValue* property. Instead, the data stored in the *MinValue* property is scaled data that has not been unit-converted (similar to the data stored in the *Value* property). As a result, you need to consider the unit conversions applied by the OpenLNS Object Server when you write to the *FormattedValue* property to avoid violating the range established for the data point. You can determine what unit conversions are applied to a particular data point by reading the *UnitsMultiplier* and *UnitsAdder* properties of the *FormatSpec* object the data point is using.

Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*minValue* = *dataPoint.SourceIndex* **Element** **Description** *minValue* The minimum value that can be applied to the *FormattedValue* property. *dataPoint* The *DataPoint* object being acted upon.
Data Type	Double.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

RawValue

Summary	Contains the value of the *DataPoint* as a raw byte array. To read the value of the data point's source object (the value of the source object in the OpenLNS database and on the network device containing the source object) as a raw byte array, call the *DataPoint* object's *Read* method, and then read this property. If the *AutoRead* property is set to True, the *Read* method will be called automatically each time this property is read. To write the value stored in this property to the data point's source object, set this property to the desired value, and call the DataPoint object's *Write* method. If the *AutoWrite* property is set to True, the *Write* method will automatically be called each time this property is written. You should note that this property shares memory space with the *Value* and *FormattedValue* properties, meaning that the information contained in all three properties represents the same value. The only difference between the three properties is that each of them is formatted differently. So, when you write to the *RawValue* property, OpenLNS will update the values of the *Value* and *FormattedValue* properties automatically, to match the updated *RawValue* property. If the data point's source object is a network variable, you should note that reading the *RawValue* property always causes a poll and returns the resulting response, even if the network variable is bound to the host or if the Data Server is already polling. If the *RawValue* property is read, and the poll fails, an exception will be thrown. If the data point's source object is a configuration property, you can use the ValueStatus property to determine if the value stored in the OpenLNS database for the configuration property matches the value stored on the physical device. In addition, the setting of the DataPoint object's SourceOptions property affects how LNS accesses the raw value of the configuration property (either from the physical device containing the configuration property, or from the LNS database).You should also be aware that some configuration properties contain a constant attribute. If a configuration property has the constant attribute set, it marks the configuration property as read-only. If this attribute is set and you attempt to write to the *RawValue* property, an exception will be thrown. However, in some cases, you can use the ConstantAttribute property to set or turn off this attribute using LNS. You can use the *MinValue* and *MaxValue* properties to set the minimum and maximum possible values for the data point. However, OpenLNS will only enforce the range established by these properties when you write to the *FormattedValue* property. You should program your application to enforce this range for the *RawValue* property

	to avoid passing invalid data to data point.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*value* = *dpObject.RawValue* **Element** — **Description** *value* — The value of the data point as a raw byte array. *dpObject* — The *DataPoint* object to be acted on.
Data Type	Variant.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

SourceIndex

Summary	Each data point that is acquired through a *ConfigProperty* object that represents arrays of elements must apply to a single element within that array. This property applies mainly to *DataPoint* objects that are acquired through *ConfigProperty* objects. Some *ConfigProperty* objects represent arrays of elements. In this case, you need to create a separate data point to read and write to each element in the array. When you create a *DataPoint* object for such *ConfigProperty* objects with the *GetDataPoint* method, you specify the index number of the element to which the new *DataPoint* object should apply. This property returns that index value. You can write to this value if you want the *DataPoint* object to apply to a different element of the *ConfigProperty* array. These arrays are 0-based, and the maximum index value is specified by the *Dimension* property of the source *ConfigProperty* object. You may find this procedure simpler than creating a separate data point for each element in the array. For *DataPoint* objects that were not acquired through *ConfigProperty* objects, this property will be initialized to 0, and you should not write to it.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*index* = *dataPoint*.**SourceIndex** **Element** — **Description** *index* — The index number assigned to the element associated with the data point. *dpObject* — The *DataPoint* object to be acted on.

Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

SourceOptions

Summary	You can use this property to determine which options were used when the *DataPoint* was created. In doing so, you can determine what information will be returned or altered when you read or write to the value of that *DataPoint*. This property is most useful for DataPoints that were acquired through *ConfigProperty* objects. You can create such DataPoints with the *GetDataPoint* method. When you do so, you will use the options element to specify how OpenLNS will reconcile differences between the value of the *ConfigProperty* in the OpenLNS database, and on the physical device, when you read or write to the value of the DataPoint. You can read this property to determine which options were used when the *DataPoint* was created, and in doing so determine what information will be returned or altered when you read or write to the value of that *DataPoint*. You can also write to the *SourceOptions* property to change the behavior of the *DataPoint* when you write to its value. If the *DataPoint* was not created for use with a *ConfigProperty* object, this property will return the **lcaDataSourceOptionsNotACp** (-1) value. In this case, the property is read-only.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*options* = *dpObject*.**SourceOptions** **Element** — **Description** *options* — A Long value indicating which options were used when the DataPoint was created. The possible values, which are stored in the *ConstDataSourceOptions* constant, are as follows: **0 lcaDataSourceOptionsNormal** The value of the *ConfigProperty* will be updated in the OpenLNS database and in the physical device each time you write to the value of the *DataPoint*. If the source *ConfigProperty* is device-specific, the value will be read directly from the device when you read the value of the *DataPoint*. If the source *ConfigProperty* is not

		device-specific, the value will be read from the database, as long as it is stored there. If its value does not exist in the database, then the value will read directly from the device, as long as the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0)**. If the source *ConfigProperty* is not device-specific, the value is not in the database, and the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**, then an exception will be thrown when you read the value of the *DataPoint*. **1 lcaDataSourceOptionsFromDevice** The data point value is always matched to the value of the source *ConfigProperty* in the physical device. You can use this information to synchronize the value of a *ConfigProperty* in the OpenLNS database with the value stored in the physical device. To do so, read the value of a data point created with this option set. Then, set the SourceOptions property of the data point to **lcaDataSourceOptionsDatabaseOnly (2)**, and call the *Write* method. The value of the source *ConfigProperty* in the OpenLNS database will then match the value of the configuration property on the physical device. **2 lcaDataSourceOptionsDatabaseOnly** The data point value is always read from the OpenLNS database. When you write to the data point, the new value will be written to the *ConfigProperty* in the OpenLNS database only, and not to the physical device. Writing to a *DataPoint* with the SourceOptions property set to this value is recommended only when updating the database with a value that has just been read from the device. For more information on this, see the description of the **lcaDataSourceOptionsFromDevice (1)** value. If you read the value of the data point, and the value does not exist in the OpenLNS database, then the **NS, #113 lcaErrNsCpValueNotFound** exception is

	thrown. **3 lcaDataSourceOptionsTypeDefaultValue** The data point value is set to the default value of configuration properties using the same type as the source configuration property. The default value is generally read from the functional profile template on the device containing the configuration property, or from the type definition for this configuration property type. Data points created with this option set are read-only. Note that this value represents the "type default", as defined in the resource files. The default value of a given configuration property may differ from the default value of its type, since the default configuration property values for a given template are defined in the external interface file and can be set from the current values in the device. *dpObject* The *DataPoint* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

TypeName

Summary	Contains the name of the current type used by this data point. This type is defined by the data point's *FormatSpec* property. You can change the data point's type by writing to the *FormatSpec* property.
Availability	Local, full, and lightweight clients.
Syntax	*typeName* = *dpObject*.**TypeName** **Element** **Description** *typeName* The name of the type. *dpObject* The *DataPoint* object to be acted on.
Data Type	String.
Read/Write	Read-only.
Added to API	LNS Release 3.0.

Value

Summary	Contains the value of the data point as a scaled double float value. This property can be used to read and write to scalar data points.

	To read the value of the data point's source object (the value of the source object in the OpenLNS database and on the physical device containing the source object) as an unformatted numeric value, use the DataPoint object's *Read* method, and then read this property. If the *AutoRead* property is set to TRUE, the *Read* method will automatically be called each time this property is read. To write to the value stored in this property to the data point's source object, set this property to the desired value, and call the DataPoint object's *Write* method. If the *AutoWrite* property is set to TRUE, the *Write* method will automatically be called each time this property is written. You should note that this property shares memory space with the *FormattedValue* and *RawValue* properties, meaning that the information contained in all three properties represents the same value. The only difference between the three properties is that each of them is formatted differently; therefore, when you write to the *Value* property, OpenLNS will update the values of the *FormattedValue* and *RawValue* properties automatically, to match the updated *Value* property. However, you can only use the *Value* property to read and write the values of data points and data point fields that take single, scalar values, such as **SNVT_lev_percent**. To read and write the value of data points containing structures, such as **SNVT_switch**, use the *FormattedValue* and *RawValue* properties. You can only write scalar values to the *Value* property. If you write a non-scalar value to the *Value* property, the **LCA, #122** exception is thrown. If the data point's source object is a network variable, you should note that reading the *Value* property always causes a poll and returns the resulting response, even if the network variable is bound to the host or if the Data Server is already polling. If the *Value* property is read, and the poll fails, a **DS 200** exception is raised, which identifies the error as a communications error. If the data point's source object is a configuration property, you can use the *ValueStatus* property to determine if the value stored in the OpenLNS database for the configuration property matches the value stored on the physical device. In addition, the setting of the DataPoint object's *SourceOptions* property affects how OpenLNS accesses the value of the configuration property (either from the physical device containing the configuration property, or from the OpenLNS database). You should also consider that some configuration properties contain a *ConstantAttribute* property. If a configuration property has the constant attribute set, it marks the configuration property as read-only. If this attribute is set and you attempt to write to the *Value* property, an exception will be thrown. However, in some cases, you can use the *ConstantAttribute* property to set or

	turn off this attribute using OpenLNS. You can use the *MinValue* and *MaxValue* properties to set the minimum and maximum possible values for the data point. However, OpenLNS will only enforce the range established by these properties when you write to the *FormattedValue* property. You should program your application to enforce this range for the Value property to avoid passing invalid data to data point.
Availability	Local, full, lightweight, and independent clients.
Syntax	*value* = *dpObject*.**Value** **Element** **Description** *value* The name of the type. *dpObject* The *DataPoint* object to be acted on.
Data Type	Double.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

DataValue

The *DataValue* object represents a data value for a *NetworkVariable* object that is currently being monitored. It is returned by the *OnNetworkVariableUpdate Event*. Only the *AppDevice*, *MonitorTag*, and *NetworkVariable* properties are for external use; all other properties are for internal use only.

Description	A data value for a *NetworkVariable* object that is currently being monitored.
Added to API	Prior to LNS 3.0.
Accessed Through	None.
Default Property	None.
Methods	None.
Properties	• *AppDevice* • *ClassId* • *Data* • *DataType* • *FloatValue* • *IntValue* • *Length* • *MonitorTag* • *NetworkVariable* • *SourceNodeId* • *Source*SubnetId • *StringValue*

Methods

The *DataValue* object does not contain any methods.

Properties

The *DataValue* object contains the following properties:

- *AppDevice*
- *ClassId*
- *Data*
- *DataType*
- *FloatValue*
- *IntValue*
- *Length*
- *MonitorTag*
- *NetworkVariable*
- *SourceNodeId*
- *SourceSubnetId*
- *StringValue*

AppDevice

Summary	Identifies the application device that is the source of the network variable update represented by this *DataValue* object. The *DataValue* object represents a value from a network variable that is being monitored. It is returned by the *OnNetworkVariableUpdate Event*. The *AppDevice* object identifies the device containing the network variable. **Note**: OpenLNS needs to fetch this object from the OpenLNS database when you read it. As a result, you should avoid reading this property if you are writing a high-performance monitoring application.
Availability	Local, full, and lightweight clients.
Syntax	*appDevObject = dvObject.AppDevice* **Element** — **Description** *appDevObject* — The *AppDevice* object to be returned. *dvObject* — The *DataValue* object to be acted on.
Data Type	*AppDevice* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *DataPoint* object in the *ConstClassIds* constant: **49 lcaClassIdDataValue** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Data

Summary	INTERNAL USE ONLY
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DataType

Summary	INTERNAL USE ONLY
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FloatValue

Summary	INTERNAL USE ONLY
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

IntValue

Summary	INTERNAL USE ONLY
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Length

Summary	INTERNAL USE ONLY
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MonitorTag

Summary	Contains the monitor tag value assigned to the network variable via its DsMonitorTag property.
Availability	Local, full, and lightweight clients.
Syntax	*tagValue* = *dvObject*.**MonitorTag** **Element** — **Description** *dvObject* — The *DataValue* object to be acted on. *tagValue* — The value of the monitor tag as a long.
Data Type	Long
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NetworkVariable

Summary	Contains the *NetworkVariable* object associated with the specified *DataValue* object. This identifies the network variable being monitored and whose monitored value is represented by the *DataValue* object.
Availability	Local, full, and lightweight clients.
Syntax	*netVarObject* = *dvObject*.**NetworkVariable** **Element** — **Description** *dvObject* — The *DataValue* object to be acted on. *netVarObject* — The *NetworkVariable* object to be returned.
Data Type	*NetworkVariable* object
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

SourceNodeId

Summary	Indicates the node ID of the device making a network variable update to a host network variable. This value is only meaningful if the network variable being monitored is a host network variable. This property, along with the *SourceSubnetId* property, allows a number of devices on the network to each have a network variable explicitly bound to a single network variable on the host. Once the network variable has been bound and monitoring is turned on using the *MonitorTag* property, *OnNetworkVariableUpdate* events will indicate that the host network variable was updated. The *SourceSubnetId* and *SourceNodeId* properties can then be used to determine which of the devices on the network updated the host network variable. For high performance when using monitor and control, you can use a temporary or permanent monitor point with a defined *tag* for each remote network variable. You could use the tag to identifiy the network variable source directly, without having to translate the source subnet/node ID.
Availability	Local, full, and lightweight clients.
Syntax	*nodeId* = *dvObject*.**SourceNodeId** **Element** — **Description** *nodeId* — The node ID of the device which sent a network variable update. *dvObject* — The *DataValue* object to be acted on.
Data Type	Integer
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

SourceSubnetId

Summary	Indicates the subnet ID of the device making a network variable update to a host network variable. This value is only meaningful if the network variable being monitored is a host network variable. This property, along with the *SourceNodeId* property, allows a number of devices on the network to each have a network variable explicitly bound to a single network variable on the host. Once the network variable has been bound and monitoring is turned on using the *MonitorTag* property, *OnNetworkVariableUpdate* events will indicate that the host network variable was updated. The *SourceSubnetId* and *SourceNodeId* properties can then be used to determine which of the devices on the network updated the host network variable. For high performance when using monitor and control, you can use a permanent or temporary monitor point with a defined *tag* for each remote network variable. You could use the tag to identifiy the network variable source directly, without having to translate the source subnet/node ID.
Availability	Local, full, and lightweight clients.
Syntax	*subnetId* = *dvObject*.**SourceSubnetId** **Element** — **Description** *subnetId* — The subnet ID of the device which sent a network variable update. *dvObject* — The *DataValue* object to be acted on.
Data Type	Integer
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

StringValue

Summary	INTERNAL USE ONLY
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DetailInfo

The *DetailInfo* object contains an error log and communications status information for an *AppDevice* or *RouterSide*. The following table summarizes the *DetailInfo* object.

Description	An error log and communications status information for an *AppDevice* or *RouterSide*.
Added to API	Prior to LNS 3.0.

Accessed Through	*AppDevice* object *RouterSide* object
Default Property	None.
Methods	None.
Properties	• *ClassId* • *ErrorLog* • *LostMessage* • *MissedMessages* • *ModelNumber* • *Parent* • *ReceiveTxFull* • *ResetCause* • *State* • *TransactionTimeouts* • *VersionNumber* • *XmitErrors*

Methods

The *DetailInfo* object does not contain any methods.

Properties

The *DetailInfo* object contains the following properties:

- *ClassId*
- *ErrorLog*
- *LostMessage*
- *MissedMessages*
- *ModelNumber*
- *Parent*
- *ReceiveTxFull*
- *ResetCause*
- *State*
- *TransactionTimeouts*
- *VersionNumber*
- *XmitErrors*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description**

	classIdValue	The object class of the object. The following value is defined for the *DataPoint* object in the *ConstClassIds* constant: **48 lcaClassIdDetailInfo**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

ErrorLog

Summary	The most recently logged firmware error.	
Availability	Local, full, and lightweight clients.	
Syntax	*errorValue* = *detailInfoObject*.**ErrorLog**	
	Element	**Description**
	errorValue	The error code for the most recently logged error. • 0 indicates no error. • Error codes 128 and greater are reserved by the system firmware. See the *Neuron Tools Errors Guide* for the list of firmware errors and their meanings. • Error codes 1 through 127 indicate an error logged by the device's application program, and the meaning of rach of these error codes is application-specific.
	detailInfoObject	The *DetailInfo* object from which to get the information.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

LostMessages

Summary	Contains the number of times a packet was received and no application input buffer was available. This data is available for application devices and router sides.
Availability	Local, full, and lightweight clients.
Syntax	*numMessages* = *detailInfoObject*.**LostMessages**

	Element	**Description**
	numMessages	The number of lost messages.
	detailInfoObject	The *DetailInfo* object from which to get the information.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

MissedMessages

Summary	Contains the number of times a packet was received and no network input buffer was available. This data is available for application devices and router sides.	
Availability	Local, full, and lightweight clients.	
Syntax	*numMessages* = *object*.**MissedMessages**	
	Element	**Description**
	numMessages	The number of missed messages.
	object	The *DetailInfo* object from which to get the information.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ModelNumber

Summary	Contains the type of Neuron Chip in the device. This data is available for application devices and router sides. This property is used only by the LCA Field Compiler.	
Availability	Local, full, and lightweight clients.	
Syntax	*modelNum* = object.**ModelNumber**	
	Element	**Description**
	modelNum	The model number of the device's Neuron Chip. The possible values for this property, which are contained in the *ConstNeuronModels* constant, are as follows: **0 lcaNeuronModel3150** **1 lcaNeuronModelPL3150** **2 lcaNeuronModelCY3250** **8 lcaNeuronModel3120** **9 lcaNeuronModel3120E1** **10 lcaNeuronModel3120E2** **11 lcaNeuronModel3120E3** **12 lcaNeuronModel3120A20**

	13 lcaNeuronModel3120E5 **14 lcaNeuronModel3120E4** **15 lcaNeuronModelPL3120E4** **16 lcaNeuronModelCY7C53120** **32 lcaNeuronModelFT5000** **33 lcaNeuronModel 5000** **128 lcaNonNeuronModelGeneric** **129 lcaNonNeuronModelPentagon** **130 lcaNonNeuronModelMIPS** *object* The *DetailInfo* object from which to get the information.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ReceiveTxFull

Summary	Contains the number of times the device or router received a packet for which there was no more room in the device's transaction database. This data is available for application devices and router sides.
Availability	Local, full, and lightweight clients.
Syntax	*packetCount* = object.**ReceiveTxFull** **Element** **Description** *packetCount* The number of packets.

	object	The *DetailInfo* object from which to get the information.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ResetCause

Summary	Contains the cause of the most recent Neuron Chip reset. This data is available for application devices and router sides.	
Availability	Local, full, and lightweight clients.	
Syntax	*resetType* = *detailInfoObject*.**ResetCause**	
	Element	**Description**
	resetType	The type of reset. The enumerated values, which are contained in the *ConstResetCauses* constant accessible through the Visual Basic Object Browser, are as follows: **0 lcaResetNone** No reset has occurred since the last time this status was cleared. **256 (0x100) lcaResetPowerup** The last reset was done during a powerup. **257 (0x101) lcaResetHardware** The last reset was caused by activation of the reset pin. **258 (0x102) lcaResetWdt** The last reset was caused by a watch dog timer timeout. **259 (0x103) lcaResetSoftware** The last reset was caused by software.
	detailInfoObject	The *DetailInfo* object from which to get the information.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

State

Summary	Describes the state of a device, router side, router, or system. This data is available for application devices and router sides. **Setting the Device State** You cannot set the state of a device in the same transaction as an *Add*, *Commission*, or *Replace* method. The state change may fail if the device has not been completely updated due to a previous communication problem or if the device has not been commissioned. If a failure occurs, use the *Commission* method to force an update of the device. You cannot set the state of a device during the definition phase of the predefined components installation scenario. **Offline Devices** Offline devices cannot receive network events related to monitor and control. For example, if the *State* property of an application device installed on the network is set to offline, then that device will not be able to receive incoming network variable events. Additionally, devices are set offline while they are being configured or commissioned. For example, if you add or remove a connection between an application device and the Network Service Device, both the application device and Network Service Device will be set offline while the change is being made. During that time, the Network Service Device will not process network variable updates, nor will it poll network variables because the configuration of the application device and the configuration of the Network Service Device are in a state of fluctuation. However, you can use the *OnSessionChangeEvent* event to track when the Network Service Device goes online or offline.
Availability	Local, full, and lightweight clients.
Syntax	*stateValue* **=** *object*.**State** **Element** — **Description** *stateValue* — The state of the device. The enumerated values for this element, which are contained in the *ConstDeviceStates* constant, are as follows: **2 lcaStateUncnfg** The application is loaded but the configuration is either not loaded, being reloaded, or deemed corrupted due to a configuration checksum error. A Neuron Chip also can make itself unconfigured by calling the Neuron C

		function **go_unconfigured()**. The device's service LED flashes at a one second rate in this state. **3 lcaStateNoApplUncnfg** No application is loaded yet, the application is in the process of being loaded, or the application has been deemed corrupted due to an application checksum error or signature inconsistency. The application does not run in this state. The device's service LED is steadily on in this state. **4 lcaStateCnfgOnline** Normal device state. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The device's service LED is off in this state. **6 lcaStateCnfgOffline** Application loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The device's service LED is off in this state. **12 lcaStateSoftOffline** The device has an application, is configured, and is soft-offline. It will go online when it is reset or when requested to go online. The device's service LED is off in this state. **140 lcaStateCnfgBypass** The application confirmed the offline request, but is still running (bypass mode). The device's service LED is off in this state.
	object	The object to be acted on.
Data Type	Long.	

Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TransactionTimeouts

Summary	Contains the number of times an acknowledged or request/response message failed after all the retries. This data is available for application devices and router sides.
Availability	Local, full, and lightweight clients.
Syntax	*messageCount* = *detailInfoObject*.**TransactionTimeouts** **Element** — **Description** *messageCount* — The number of failed messages. *detailInfoObject* — The *DetailInfo* object from which to get the information.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

VersionNumber

Summary	Contains the version number of Neuron Chip firmware contained in a device. This data is available for application devices and router sides.
Availability	Local, full, and lightweight clients.
Syntax	*versionValue* = *detailInfoObject*.**VersionNumber** **Element** — **Description** *versionValue* — The firmware version number. This value is a decimal number from 1 to 255 indicating the version of the Neuron Chip firmware. *detailInfoObject* — The *DetailInfo* object from which to get the information.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

XmitErrors

Summary	Contains the number of times the device either received a packet with an invalid CRC, received a packet that was too short, received a packet too long for the device's inpurt buffer, or saw a timeout. This data is available for application devices and router sides.
Availability	Local, full, and lightweight clients.

Syntax	*errorCount* = *detailInfoObject*.**XmitErrors** **Element** **Description** *errorCount* The number of errors. *detailInfoObject* The *DetailInfo* object from which to get the information.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DeviceTemplate

A *DeviceTemplate* object represents generic device type information that can be applied to multiple LONWORKS application devices.

The deprecated *Export* and *Link* methods and the deprecated *BuildStatus*, *ExportFormat* and *ProgramTemplate* properties apply only if the LCA Field Compiler is used. These methods and properties will only be used if the *ProgramType* property is **lcaProgramTypeSource**.

Description	Device type information that can be applied to application devices.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object. *DeviceTemplates* collection object.
Default Property	*Name*
Methods	• *Import* • *ResyncToResources*
Properties	• *AppDevices* • *BitmapFilePath* • *ClassId* • *ComponentApps* • *Description* • *DeviceClass* • *DeviceSubclass* • *DeviceValidation* • *DynamicNvSupported* • *Extensions* • *Format* • *Handle* • *HostSelect* • *IconFilePath* • *Interface* • *ManufacturerId* • *ModelNo* • *Name* • *Parent* • *ProgramId* • *ProgramType*

	• *RegisteredComponent* • *SelfDocConsistency* • *SelfDocumentation* • *UserTypeFileName* • *XifPath*

Methods

The *DeviceTemplate* object contains the following methods:

- *Import*
- *ResyncToResources*

Import

Summary	Import an external interface from an XIF or XFB file into a device template. This updates the property values of the device template with the information from the imported external interface. The XFB file must be version 3.0 or later. If you commission a device without an imported external interface, which will cause the interface to be uploaded from the device, you can invoke this method later to refresh the external interface and incorporate the additional information provided by the XIF file. After you use the *Import* method on a *DeviceTemplate* , you should call the *ResyncToTemplate* method on all of the *AppDevice* objects using the *DeviceTemplate* to resynchronize those devices with the updated information.
Availability	Local, full, and lightweight clients.
Syntax	*devTemplateObject*.**Import** *xifPath* **Element** — **Description** *devTemplateObject* — The *DeviceTemplate* object being acted upon. *xifPath* — A String specifying the path of the XIF and XFB files to be imported.
Added to API	LNS Release 3.0.

ResyncToResources

Summary	Resychronizes the *DeviceTemplate* with modified or newly accessible device resource file information. You may need to resynchronize a *DeviceTemplate* if you imported a device's XIF, and the resource files for that device were not available in the resource file catalog at that time. In this case, some of the information contained in the *DeviceTemplate* , including the formatting of its configuration properties, may be invalid. You could also use this method if the device resource files

<table>
<tr><td></td><td>have been updated or modified since the device's XIF was imported. This method allows you to update the DeviceTemplate whenever these situations occur, without having to re-import the XIF.

The LdrfCatalogPath property points to the location of the device resource file catalog. You can browse the contents of this catalog with the NodeBuilder Resource Editor included with the Echelon device development software.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>devTemplateObject.ResyncToResources options

Element — Description

devTemplateObject — The DeviceTemplate object being acted upon.

options — Determines whether the changes to the DeviceTemplate will be automatically propagated to the devices using the template.

The possible values for this element, which are stored in the ConstResyncToResourcesOptionFlags constant, are as follows:

0
lcaResyncToResourcesOptionDefault

Does not automatically propagate DeviceTemplate changes to the devices using that DeviceTemplate

If you select this value, you can call the ResyncToTemplate method later on any application devices that are using the DeviceTemplate to propagate the changes.

16,777,216
lcaResyncToResourcesOption
UpdateDevices

Automatically propagates DeviceTemplate changes to the devices using that DeviceTemplate .

If you select this value, you can OR it with any of the enumerations of the ConstResyncToTemplateOptionFlags constant to determine whether the names of the configuration properties, network variables, and LonMarkObjects on the devices should be updated as part of the</td></tr>
</table>

	resynchronization.
Added to API	LNS Release 3.20.

Properties

The *DeviceTemplate* object contains the following properties:

- *AppDevices*
- *BitmapFilePath*
- *ClassId*
- *ComponentApps*
- *Description*
- *DeviceClass*
- *DeviceSubclass*
- *DeviceValidation*
- *DynamicNvSupported*
- *Extensions*
- *Format*
- *Handle*
- *HostSelect*
- *IconFilePath*
- *Interface*
- *ManufacturerId*
- *ModelNo*
- *Name*
- *Parent*
- *ProgramId*
- *ProgramType*
- *RegisteredComponent*
- *SelfDocConsistency*
- *SelfDocumentation*
- *UserTypeFileName*
- *XifPath*

AppDevices

Summary	Contains the *AppDevices* collection object associated with the specified *Channel* object. The *AppDevices* property represents all the devices on the channel.
Availability	Local, full, and lightweight clients.
Syntax	*appDevicesCollection* = *object*.**AppDevices** **Element** — **Description** *appDevicesCollection* — The *AppDevices* collection returned. *object* — The *Channel* object to be acted on.
Data Type	*AppDevices* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.
Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** — **Description** *bmpFilePath* — The bitmap path and file name. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *DataPoint* object in the *ConstClassIds* constant: **36 lcaClassIdDeviceTemplate** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.

ComponentApps

Summary	Contains the *ComponentApps* collection object associated with the specified *DeviceTemplate*. The *ComponentApps* collection is a list of LNS plug-in commands that are associated with a particular object type. Note that all *LonMarkObject* objects contain a *ComponentApps* property; however, the behavior of this property is unspecified when accessed through a *LonMarkObject* object.
Availability	Local, full, and lightweight clients.
Syntax	*appsCollection* **=** *object*.**ComponentApps** **Element** — **Description** *appsCollection* — The *ComponentApps* collection to be returned. *object* — The *DeviceTemplate* object to be acted on.
Data Type	*ComponentApps* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *DeviceTemplate* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *DeviceTemplate* object. *object* — The *DeviceTemplate* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

DeviceClass

Summary	Stores the device class for a device template.
Availability	Local, full, and lightweight clients.
Syntax	*deviceClassValue* = *devTemplateObject*.**DeviceClass** **Element** — **Description** *deviceClassValue* — The device class value *devTemplateObject* — The *DeviceTemplate* object.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DeviceSubclass

Summary	Returns the device subclass for a device template.
Availability	Local, full, and lightweight clients.
Syntax	*subClassValue* = *devTemplateObject*.**DeviceSubclass** **Element** — **Description** *subClassValue* — The device sub-class value *devTemplateObject* — The *DeviceTemplate* object.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DeviceValidation

Summary	Determines the device validation steps OpenLNS performs when commissioning, replacing or upgrading devices that use this device template. This may help reduce the time require for commissioning a large number of devices if you are confident that the devices contain the correct program information and that they are installed on the correct channel. This method may also affect the validation performed when loading a device's application image with the *Load* or *LoadEx* methods. By default, OpenLNS performs a device validation procedure when a device is commissioned with the *Commission* method and the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)**, or when a device is commissioned with the *CommissionEx* method and the **lcaCommissionFlagPropagateUpdates (1)** flag is set. This procedure will validate that the device is on the channel specified by the user, the program ID in the device matches

<table>
<tr><td></td><td>that specified by the application, and the program interface in the device matches the previously defined program interface for that program ID. This validation occurs to prevent users from configuring a device incorrectly.

This validation, however, does require the transmission of a large number of messages between OpenLNS and the device. This can be very time consuming, particularly on slow media, and can greatly increase the time required to commission the devices on a network.

You can use this property to determine which validation steps OpenLNS will perform when you are commissioning, replacing or upgrading a device.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>validationOption = devTemplateObject.DeviceValidation

Element — Description

validationOption — The validation options used when commissioning or replacing devices that use this device template.

The valid settings for this property, which are contained in the ConstDeviceValidation constant, are as follows:

0 lcaDeviceValidationNormal

This is the default value. When this value is used, all validation steps will be performed.

In this case, the commission procedure will validate that the physical device has the same external interface and program ID as defined for the AppDevice object in the OpenLNS database. It will also validate that the device is on the channel assigned to it in the OpenLNS database.

If the physical device is not using the same external interface or program ID as defined for the AppDevice object in the database, the commission will fail, and either the NS, #59 lcaErrNsProgramIntfMismatch or NS, #38 lcaErrNsProgramidMismatch exceptions will be thrown.

If the physical device is not on the channel assigned to it in the</td></tr>
</table>

		database, the commission will fail, and the **NS, #72 lcaErrNsWrongChannel** exception will be thrown. **1 lcaDeviceValidationNoChannelValidation** Do not validate the channel if it is already known because either it has been specified by the application, or the device is already registered. If this option is selected and the channel is not specified, the channel isolation process will still be used. If this option is selected and a device has been placed on the wrong channel, OpenLNS will not detect this. As a result, communication with the device may not work from some channels due to routing constraints. In addition, layer 4 timers may be set incorrectly, which will cause communication problems for the device. Note that if the program ID of a device is not known, OpenLNS must read the device's program ID before commissioning, upgrading, or replacing the device. Before reading the program ID, OpenLNS will perform channel validation, regardless of whether this flag is set. As a result, this value has no effect if the program ID of the device is not known. **2 lcaDeviceValidationNoProgramInterfaceValidation** Do not validate the program interface. If the program interface is not known, it will still be recovered if this option is set. When loading the application image for a device that has already been commissioned, OpenLNS reads the program ID from the newly loaded device, and if it matches the original, it validates the program interface. This step will be skipped if this

		option is set. Note that the **lcaDeviceValidationNoChannelValidation (1)** and **lcaDeviceValidationNoProgramIdValidation (4)** flags have no effect during an application download. If this option is set and a device is using a program interface that is inconsistent with the device template, the device will be configured incorrectly, and unpredictable behavior will occur. In addition, device corruption may prevent the device from being loaded or upgraded in the future. **4 lcaDeviceValidationNoProgramIdValidation** Do not validate the program ID of the device if it has been specified by the application or it is already known, because the device has already been registered. If the program ID is not known, it will still be recovered from the device if this option is set. If this option is set, the program interface will not be validated either. If this option is set and a device is using a program interface that is inconsistent with the device template, the device will be configured incorrectly and unpredictable behavior will occur. In addition, device corruption may prevent the device from being loaded or upgraded in the future.
	devTemplateObject	The *DeviceTemplate* object to be acted upon.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

DynamicNvSupported

Summary	Specifies whether devices created with this device template support dynamic network variables.

Availability	Local, full, and lightweight clients.
Syntax	*dynamicNvSupValue* = *dtObject*.**DynamicNvSupported** **Element** — **Description** *dynamicNvSupValue* — A value of 0 or 1. **0** Devices based on this device template do not support dynamic network variables. **1** Devices based on this device template support network variables. *dtObject* — The *DeviceTemplate* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *DeviceTemplate* . This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer.
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.**Extensions** **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Format

Summary	Stores program ID type information for a device template.
Availability	Local, full, and lightweight clients.
Syntax	*formatValue* = *devTemplateObject*.**Format** **Element** — **Description** *formatValue* — The program ID descriptor. This element may be one of the following

	values: **0** Development program ID. **8** LonMark approved program ID. **9** LonMark prototype program ID. *devTemplateObject* The *DeviceTemplate* object
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *DeviceTemplate* object. This property enables the *DeviceTemplate* to be retrieved by the *ItemByHandle* method of the *DeviceTemplates* object.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *DeviceTemplate* .**Handle** **Element** **Description** *returnValue* The NSS handle of the *DeviceTemplate* object. *object* The *DeviceTemplate* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	OpenLNS.

HostSelect

Summary	Indicates whether the network variable processing on devices using this device template is managed on the Neuron Chip or on an attached processor. The Microprocessor Interface Program (MIP) is firmware for the Neuron Chip that transforms the Neuron Chip into a communications co-processor for an attached host processor. For more information on the MIP firmware, see the LONWORKS Microprocessor Interface Program (MIP) User's Guide. This can be downloaded from Echelon's website at *www.echelon.com*.
Availability	Local, full, and lightweight clients.
Syntax	*hostSelect* = *dtObject*.**HostSelect** **Element** **Description** *hostSelect* A Boolean value. **TRUE**. Device is a host-based

	device. **FALSE**. Device is a Neuron hosted device. *dtObject* The *DeviceTemplate* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (***.ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** **Description** *IconFilePathFileName* Icon file and path name *object* The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

Interface

Summary	Contains the main *Interface* object associated with the specified *AppDevice* object. This includes the static interface of the device, as well as all custom, virtual interfaces that have been added to the device dynamically, with the *Add* method. The collection of custom interfaces that have been added to a device is contained in the *Interfaces* property.
Availability	Local, full, and lightweight clients.
Syntax	*interfaceObject* = *object*.**Interface** **Element** **Description** *interfaceObject* The *Interface* object retrieved from the object.

	object The *AppDevice* object to be acted on.
Data Type	*Interface* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ManufacturerID

Summary	Reads the manufacturer ID assigned to a device template or component application. This property is the manufacturer ID field of the standard program ID. See the *LonMark Application Layer Interoperability Guidelines* for more information.
Availability	Local, full, lightweight, and independent clients.
Syntax	*idValue* = *object*.**ManufacturerId** **Element** **Description** *idValue* The component application or device template's manufacturer ID. *object* The *ComponentApp* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ModelNo

Summary	Reads the model number for a device template.
Availability	Local, full, and lightweight clients.
Syntax	*modelValue* = *templateObject*.**ModelNo** **Element** **Description** *modelValue* The model number, which is the model number field of the standard program ID. *templateObject* The *DeviceTemplate* object.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it

	may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ProgramId

Summary	Stores the program ID for the device template. Every LonMark compliant LONWORKS application device uses a unique, 16 digit, hexadecimal standard program ID that uses the following format: FM:MM:MM:CC:CC:UU:TT:NN See the *Devices Interfaces* section in the *OpenLNS Programmer's Guide* for a description of the format used to display program IDs.

Availability	Local, full, and lightweight clients.
Syntax	*programIdValue* = *object*.**ProgramId** **Element** — **Description** *programIdValue* — The program ID value of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ProgramType

Summary	Identifies the form or origin of the application program for this device. If this property is set to **lcaProgramType**, the deprecated *ProgramTemplate* property of the *DeviceTemplate* object specifies a valid *ProgramTemplate* object.
Availability	Local, full, and lightweight clients.
Syntax	*programTypeValue* = *devTemplateObject*.**ProgramType** **Element** — **Description** *programTypeValue* — The device template's program type. The values for this element, which are contained in the *ConstProgramTypes* constant, are as follows: **0 lcaProgramTypeSource** The application program is defined by a Neuron C source file. **1 lcaProgramTypeXif** The application program is defined by an external interface (.XIF extension) file. **2 lcaProgramTypeFromDevice** This value is not supported. *devTemplateObject* — The *DeviceTemplate* object.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

RegisteredComponent

Summary	Contains the ActiveX name for the device control used by this device template.

	Each *DeviceTemplate* object in an OpenLNS database may have a single device control registered for it. A null value in this property indicates that there is no registered device control for this *DeviceTemplate* object. To support device controls, a director application must follow these steps when creating a new device: 1. Examine the *RegisteredComponent* property of the *DeviceTemplate* object for the new device. If a device control is registered, create an instance of the control using the ActiveX name in the *RegisteredComponent* property. 2. Assign the instance of the control created to the deprecated *ActiveXComponent* property of the new *AppDevice* object. 3. Invoke the deprecated *OpenComponent* method of the new *AppDevice* object.
Availability	Local, full, and lightweight clients.
Syntax	*regComp* = *dtObject*.**RegisteredComponent** **Element** — **Description** *regComp* — The registered component for this device template. *dtObject* — The *DeviceTemplate* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

SelfDocConsistency

Summary	Determines how much OpenLNS will assume about the self documentation of devices that are using this template. This affects how OpenLNS will read the self-documentation data of those devices, and what level of program interface validation OpenLNS will perform when commissioning those devices. All Neuron hosted devices and host-based devices contain self-documentation data. Some of this data appears in a device's external interface file. OpenLNS uses a device's self-documentation data to communicate with it. The setting of this property affects how OpenLNS will read the self-identification data of devices using this template, and what level of program interface validation OpenLNS will perform when commissioning those devices.
Availability	Local, full, and lightweight clients.
Syntax	*consistencyValue* = *devTemplateObject*.**SelfDocConsistency** **Element** — **Description** *consistencyValue* — The self-documentation consistency flag assigned to devices using this device

		template. The valid values for this property, which are contained in the *ConstSelfDocConsistency* constant, are as follows: **0 lcaSelfDocConsistencyIdenticalOnAllDevices** The self-documentation on devices using the device template must be identical. This is the most efficient, but least flexible, setting of this property. It is the default for all devices without a LonMark standard program ID. **1 lcaSelfDocConsistencyStringsMayDifferByDevice** The self-documentation on devices of this type may include different self-documentation strings, but must use the same self-information data format. This is the default for all devices with a LonMark standard program ID. **2 lcaSelfDocConsistencyStringsAndFormatMayDifferByDevice** The self-documentation on devices of this type may include different self-documentation strings, and may use different self-documentation data formats. The more restrictive values (**lcaSelfDocConsistencyIdenticalOnAllDevices 0** or **lcaSelfDocConsistencyStringsMayDifferByDevice 1)** allow OpenLNS to assume higher degrees of self-documentation consistency among devices using this template, and cause OpenLNS to operate more efficiently when validating the program interface of those devices while commissioning. However, these settings may cause problems if the device developer has produced multiple devices that have the same program ID, but use different self-documentation strings or formats. As a result, do not set this property to a

		value that conflicts with the configurations of the devices using a template. If you set this property to a setting that violates the configuration of devices already using the template, then the **NS, #59 lcaErrNsProgramIntfMismatch** exception is thrown.
	devTemplateObject	The *DeviceTemplate* object to be acted upon.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

SelfDocumentation

Summary	Stores the self-documentation string of the application device. The length of the string is not provided as a separate property. To get the length, get the *descriptionString*, and calculate the length from it. Note that this property returns only the user portion (which follows the LonMark portion, if any) of the self-documentation string	
Availability	Local, full, and lightweight clients.	
Syntax	*descriptionString* = *object*.**SelfDocumentation**	
	Element	**Description**
	programIdValue	The program ID value of the object.
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

UserTypeFileName

Summary	Contains an alternative base file name for user formatting. Setting this property to "<my fomat file>" causes the Object Server to automatically prepend "<my format file>." when assigning the network variable's type. Otherwise, it will defer to the default Data Server behavior.	
Availability	Local, full, and lightweight clients.	
Syntax	*formatString* = *devTemplateObject*.**UserTypeFileName**	
	Element	**Description**
	formatString	The user type format file path. If you specify this element, the string value will be used as the

		base name of the formatting file (**.fmt**) instead of the default.
	devTemplateObject	The *DeviceTemplate* object to be acted on.
Data Type	String.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

XifPath

Summary	Contains the path of the external interface file used by this device template.	
Availability	Local, full, and lightweight clients.	
Syntax	*xifPathValue* = *dtObject*.**XifPath**	
	Element	**Description**
	xifPathValue	The external interface file path. If the name is provided without a path, OpenLNS will look in the directory specified in the *System* object's *ImportDirectory* property.
	dtObject	The *DeviceTemplate* object from which to get the information.
Data Type	String.	
Read/Write	Read only	
Added to API	LNS Release 3.0.	

DeviceTemplates

The *DeviceTemplates* object is a collection of *DeviceTemplate* objects. The instance of this collection, which is accessed through the *TemplateLibrary* object, holds all of the *DeviceTemplate* objects in the system. The following table summarizes the *DeviceTemplates* object.

Description	Represents a collection of *DeviceTemplate* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*TemplateLibrary* object.
Default Property	*Item* property.
Methods	• *Add* • *ItemByHandle* • *ItemByProgramId* • *Remove*
Properties	• *ClassID* • *Count* • *Item*

	• *Parent* • *_NewEnum*

Methods

The *DeviceTemplates* object contains the following methods.

- *Add*
- *ItemByHandle*
- *ItemByProgramId*
- *Remove*

Add

Summary	Defines a new *DeviceTemplate* object. A *DeviceTemplate* object needs to be explicitly created with this method only if that *DeviceTemplate* is going to be initialized by importing an external interface file (.**XIF** and .**XFB** extensions). If the *DeviceTemplate* is to be initialized over the network from a device, the *DeviceTemplate* is created automatically by the Object Server (see the *Add* method of the *AppDevice* object for more information).
Availability	Local, full, and lightweight clients.
Syntax	*devTemplateObject* = *devTemplatesColl*.**Add** (*devTemplateName, programType*) **Element** — **Description** *devTemplateObject* — The newly defined *DeviceTemplate* object. *devTemplatesColl* — The *DeviceTemplates* collection object. *devTemplateName* — A String containing the name of the device template. *programType* — The type of program associated with this device template as an integer. The enumerated values for this value, which are contained in the *ConstProgramTypes* constant, are as follows: **0 lcaProgramTypeSource** The application program is defined by a Neuron C source file. **1 lcaProgramTypeXif** The application program is defined by an external interface (.XIF extension) file. **2 lcaProgramTypeFromDevice**

	This value is not supported.
Added to API	Prior to LNS Release 3.0.

ItemByHandle

Summary	Retrieves a *DeviceTemplate* object by its *handle* property from the *DeviceTemplates* collection. This may be particularly useful when processing events reported by **OnChangeEvent** with the objectType of **lcaChangeEventDeviceTemplates**.
Availability	Local, full, and lightweight clients.
Syntax	*deviceTemplate* = *deviceTemplates*.**ItemByHandle** **Element** — **Description** *deviceTemplate* — The *DeviceTemplate* object. *deviceTemplates* — The *DeviceTemplates* collection. *handle* — The handle of the *DeviceTemplate* to be retrieved.
Added to API	OpenLNS.

ItemByProgramID

Summary	Retrieves a *DeviceTemplate* object, specified by its *ProgramID* property, from a *DeviceTemplates* collection.
Availability	Local, full, and lightweight clients.
Syntax	*dtObject* = *dtColl*.**ItemByProgramId**(*programId*) **Element** — **Description** *dtObject* — The *DeviceTemplate* object retrieved from the collection. *dtColl* — The *DeviceTemplates* collection object. *programId* — A String specifying the program ID of the *DeviceTemplate* object to be retrieved.
Added to API	Prior to LNS Release 3.0.

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed.

	name	A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.	

Properties

The *DeviceTemplates* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *DeviceTemplates* object in the *ConstClassIds* constant: **37 lcaClassIdDeviceTemplates** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count**

	Element	**Description**
	returnValue	The number of objects in the collection as a long integer.
	object	The collection object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Item

Summary	Returns a *DeviceTemplate* object from the *DeviceTemplates* collection. You can retrieve a *DeviceTemplate* object from its *DeviceTemplates* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *DeviceTemplate* object in a *DeviceTemplates* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*)
	Element / **Description** *retrievedObject* — The *DeviceTemplate* object retrieved from the collection. *collObject* — The *DeviceTemplates* collection object to be acted on. *index* — A Long type specifying the ordinal index of the object to be retrieved. *stringExpression* — A string type specifying the name of the object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

Error

An *Error* object represents the information for a single Object Server error. Errors are handled by LCA applications via the ActiveX exception mechanism. The *Error* object contains the standard code and description properties that are passed back in the last exception, as well as some additional information. The following table summarizes the *Error* object.

Description	The data for a single Object Server error.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None
Methods	None
Properties	• *Category* • *ClassId* • *ConnErrAppDeviceHandle1* • *ConnErrAppDeviceHandle2* • *ConnErrIndex1* • *ConnErrIndex2* • *ConnErrIndexType* • *ConnErrIndexType2* • *Description* • *ErrObjClassId* • *ErrObjHandle* • *Number* • *Parent*

Methods

The *Error* object does not contain any methods.

Properties

The *Error* object contains the following properties:

- *Category*
- *ClassId*
- *ConnErrAppDeviceHandle1*
- *ConnErrAppDeviceHandle2*
- *ConnErrIndex1*
- *ConnErrIndex2*
- *ConnErrIndexType*
- *ConnErrIndexType2*
- *Description*
- *ErrObjClassId*
- *ErrObjHandle*
- *Number*
- *Parent*

Category

Summary	Contains the error category represented by the *Error* object.
Availability	Local, full, and lightweight clients.
Syntax	*errCategory* = e*rrorObject*.**Category** **Element** — **Description** *errCategory* — The error category. The valid values for this property, which are contained in the *ConstErrCategories* constant, are as follows: **0 lcaErrCategoryNs** Network Server error. **1 lcaErrCategoryNi** Network Interface error. **2 lcaErrCategoryConn** Connection error. **3 lcaErrCategoryObjServer** Object Server error. **4 lcaErrCategoryDataServer** Data Server error. **5 lcaErrCategoryFormat** Format error. **6 lcaErrCategoryComponent** Component error. **7 lcaErrCategoryVni** VNI error. *object* — The *Error* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Error* object in the *ConstClassIds* constant: **44 lcaClassIdError** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ConnErrAppDeviceHandle1

Summary	If the error object represents a connection subsystem error (**lcaErrCategoryConn**), this property returns the handle of the primary device in the error. Otherwise it returns 0. You can use the *AppDevices* collection object's *ItemByHandle* method to get the *AppDevice* corresponding to the device handle.
Availability	Local, full, and lightweight clients.
Syntax	*errorValue* = *errorObject*.**ConnErrAppDeviceHandle1** **Element** **Description** *deviceHandle* The device handle to be returned. *errObject* The *Error* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConnErrAppDeviceHandle2

Summary	If the *Error* object represents a connection subsystem error (**lcaErrCategoryConn**), this property returns the handle of the secondary device in the error. Otherwise it returns 0. You can use the *AppDevices* collection object's *ItemByHandle* method to get the *AppDevice* corresponding to the device handle.
Availability	Local, full, and lightweight clients.
Syntax	*errorValue* = *errorObject*.**ConnErrAppDeviceHandle2** **Element** **Description** *deviceHandle* The device handle to be returned.

	errObject The *Error* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConnErrIndex1

Summary	If the *Error* object represents a connection error (**lcaErrCategoryConn**), this property returns the index of the primary network variable or message tag in the error. Otherwise it returns 0. The *ConnErrIndexType1* property indicates whether this index applies to a network variable or message tag, as well as the type of network variable or message tag. The value of this property combined with that of *ConnErrAppDeviceHandle1* identifies a specific network variable or message tag in a specific device You can use the *AppDevices* collection object's *ItemByHandle* method to get the *AppDevice* corresponding to the device handle.
Availability	Local, full, and lightweight clients.
Syntax	*index* = *errorObject*.**ConnErrIndex1** **Element** **Description** *index* The network variable or message tag index to be returned. *errObject* The *Error* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConnErrIndex2

Summary	If the *Error* object represents a connection error (**lcaErrCategoryConn**), this property returns the index of the primary network variable or message tag in the error. Otherwise it returns 0. The *ConnErrIndexType2* property indicates whether this index applies to a network variable or message tag, as well as the type of network variable or message tag. The value of this property combined with that of *ConnErrAppDeviceHandle2* identifies a specific network variable or message tag in a specific device

	You can use the *AppDevices* collection object's *ItemByHandle* method to get the *AppDevice* corresponding to the device handle.
Availability	Local, full, and lightweight clients.
Syntax	*index* = *errorObject*.**ConnErrIndex2** **Element** **Description** *index* The network variable or message tag index to be returned. *errObject* The *Error* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConnErrIndexType1

Summary	If the *Error* object represents a connection error (**lcaErrCategoryConn**), this property indicates whether a network variable or message tag index is contained in the *ConnErrIndex1* property. The *ConnErrIndexType1* property indicates whether this index applies to a network variable or message tag, as well as the type of network variable or message tag. The value of this property combined with that of *ConnErrAppDeviceHandle1* identifies a specific network variable or message tag in a specific device You can use the *AppDevices* collection object's *ItemByHandle* method to get the *AppDevice* corresponding to the device handle.
Availability	Local, full, and lightweight clients.
Syntax	*indexType* = *errorObject*.**ConnErrIndexType1** **Element** **Description** *indexType* The type of network variable or message tag that caused the error. The enumerated values for this property, which are contained in the *ConstLNSIndexType* constant, are as follows: **0 lcaLNSIndexTypeNv** The index in the corresponding *ConnErrIndex1* property is a network variable index. **1 lcaLNSIndexTypeMessageTag** The index in the corresponding

		ConnErrIndex1 property is a message tag index. **2 lcaLNSIndexTypeDynamicMessageTag** The index in the corresponding *ConnErrIndex1* property is a dynamic message tag index. **-1 lcaLNSIndexTypeInvalid** The corresponding *ConnErrIndex* property contais invalid data.
	errObject	The *Error* object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ConnErrIndexType2

Summary	If the *Error* object represents a connection error (**lcaErrCategoryConn**), this property indicates whether a network variable or message tag index is contained in the *ConnErrIndex2* property. The *ConnErrIndexType2* property indicates whether this index applies to a network variable or message tag, as well as the type of network variable or message tag. The value of this property combined with that of *ConnErrAppDeviceHandle2* identifies a specific network variable or message tag in a specific device You can use the *AppDevices* collection object's *ItemByHandle* method to get the *AppDevice* corresponding to the device handle.	
Availability	Local, full, and lightweight clients.	
Syntax	*indexType* = *errorObject*.**ConnErrIndexType2**	
	Element	**Description**
	indexType	The type of network variable or message tag that caused the error. The enumerated values for this property, which are contained in the *ConstLNSIndexType* constant, are as follows: **0 lcaLNSIndexTypeNv** The index in the corresponding *ConnErrIndex2* property is a network variable index. **1 lcaLNSIndexTypeMessageTag**

	The index in the corresponding *ConnErrIndex2* property is a message tag index. **2 lcaLNSIndexTypeDynamicMessageTag** The index in the corresponding *ConnErrIndex2* property is a dynamic message tag index. **-1 lcaLNSIndexTypeInvalid** The corresponding *ConnErrIndex2* property contais invalid data. *errObject* The *Error* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *Error* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *Error* object. *object* The *Error* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

ErrObjClassId

Summary	Contains the *ClassId* of the object that caused the error contained in this Error object.
Availability	Local, full, and lightweight clients.
Syntax	*errorId* = *errorObject*.**ErrorObjClassId** **Element** **Description** *errorObject* The *Error* object to be acted on. *errorId* A short that indicates the ClassId of the object that caused the error. The possible values for this property, which are

	contained in the *ConstClassIds* constant, are as follows: **0 lcaClassIdObjectServer** **1 lcaClassIdNetwork** **2 lcaClassIdNetworks** **3 lcaClassIdSyste** **4 lcaClassIdSystem** **5 lcaClassIdSubsyste** **6 lcaClassIdSubsystem** **7 lcaClassIdAppDevic** **8 lcaClassIdAppDevice** **9 lcaClassIdRoute** **10 lcaClassIdRouters** **11 lcaClassIdRouterSide** **12 lcaClassIdChannel** **13 lcaClassIdChannels** **14 lcaClassIdNetworkInterface** **15 lcaClassIdNetworkInterfaces** **16 lcaClassIdSubnet** **17 lcaClassIdSubnets** **18 lcaClassIdConnections** **19 lcaClassIdInterface** **20 lcaClassIdInterfaces** **21 lcaClassIdTemplateLibrary** **22 lcaClassIdNetworkVariable** **23 lcaClassIdNetworkVariables** **24 lcaClassIdMessageTag** **25 lcaClassIdMessageTags** **26 lcaClassIdConfigProp** **27 lcaClassIdConfigProps** **28 lcaClassIdLonMarkObject** **29 lcaClassIdLonMarkObjects** **30 lcaClassIdComponentApp** **31 lcaClassIdComponentApps** **32 lcaClassIdHardwareTemplate** **33 lcaClassIdHardwareTemplates** **34 lcaClassIdBuildTemplate** **35 lcaClassIdBuildTemplates** **36 lcaClassIdDeviceTemplate** **37 lcaClassIdDeviceTemplates** **38 lcaClassIdProgramTemplate** **39 lcaClassIdProgramTemplates** **40 lcaClassIdNetworkServiceDevice** **41 lcaClassIdNetworkServiceDevices** **42 lcaClassIdConnectDescTemplate** **43 lcaClassIdConnectDescTemplates** **44 lcaClassIdError** **45 lcaClassIdLonMarkAlarm** **46 lcaClassIdObjectStatus** **47 lcaClassIdNetworkVariableField** **48 lcaClassIdDetailInfo** **49 lcaClassIdDataValue** **50 lcaClassIdExtension**

	51 lcaClassIdExtensions **52 lcaClassIdRecoveryStatus** **53 lcaClassIdCreditInfo** **54 lcaClassIdAccount** **55 lcaClassIdAccounts** **56 lcaClassIdBufferConfiguration** **57 lcaClassIdFileTransfer** **58 lcaClassIdAlias** **59 lcaClassIdAliases** **69 lcaClassIdPingIntervals** **70 lcaClassIdApplication** **71 lcaClassIdTestInfo** **72 lcaClassIdDataPoint** **73 lcaClassIdFormatSpec** **74 lcaClassIdMonitorSet** **75 lcaClassIdMonitorSets** **76 lcaClassIdMsgMonitorOptions** **77 lcaClassIdMsgMonitorPoint** **78 lcaClassIdMsgMonitorPoints** **79 lcaClassIdNvMonitorOptions** **80 lcaClassIdNvMonitorPoint** **81 lcaClassIdNvMonitorPoints** **82 lcaClassIdSourceAddress** **83 lcaClassIdLdrfLanguage** **84 lcaClassIdLdrfLanguages** **85 lcaClassIdServiceStatus** **86 lcaClassIdUpgradeStatus** **87 lcaClassIdUpgradeInfo** **88 lcaClassIdUpgradeInfos** **89 lcaClassIdDatabaseValidationReport** **90 lcaClassIdDatabaseValidationErrorSummary** **91 lcaClassIdDatabaseValidationErrorSummaries** **92 lcaClassIdDatabaseValidationErrorInstance** **93 lcaClassIdNetworkResources** **94 lcaClassIdTypeSpec** **95 lcaClassIdFormatLocale** **96 lcaClassIdFormatLocales**
Data Type	Short.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ErrObjHandle

Summary	Contains the handle of the object that caused the error contained in this Error object.
Availability	Local, full, and lightweight clients.

Syntax	*errorId* = *errorObject*.**ErrorObjHandle** **Element** **Description** *errorObject* The *Error* object to be acted on. *errorId* A short which indicates the handle of the object that caused the error. The class of object which raised the error is contained in the *ErrObjClassId* property.
Data Type	Short.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Number

Summary	Returns the LNS error number of the error represented by the Error object.
Availability	Local, full, and lightweight clients.
Syntax	*errorValue* = *errorObject*.**Number** **Element** **Description** *errorValue* The error number to be returned. The values that can be returned by this property are stored in the *ConstErrors* constant. For descriptions of these errors, see *Object Server Errors*. *errObject* The *Error* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.

Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Extension

An *Extension* object represents user-defined data. Each manufacturer can define any number of manufacturer specific data objects for application-specific data. Collections of these objects can be associated with the object server, channel, device template, program template, network, device (application device, router, or network service device), subnet, subsystem, system, or LonMarkObject. The following table summarizes the *Extension* object.

Description	User-defined data objects for application-specific data.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Extensions* collection object.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *CopyWithParent* • *Description* • *Extensions* • *Handle* • *Key* • *Owner* • *OwnerClassId* • *Parent* • *Value1* • *Value2* • *Value3*

Methods

The *Extension* object does not contain any methods.

Properties

The *Extension* object contains the following properties:

- *ClassId*
- *CopyWithParent*
- *Description*
- *Extensions*
- *Handle*
- *Key*
- *Owner*
- *OwnerClassId*
- *Parent*
- *Value1*

- *Value2*
- *Value3*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Error* object in the *ConstClassIds* constant: **50 lcaClassIdExtension** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CopyWithParent

Summary	Indicates whether the *Extension* object should be copied when its parent object is copied. This may be useful if you are writing an application that is copying an object containing a large *Extensions* collection, possibly to move the object to another database. You may not want the application to copy every extension record stored in the object you are moving. You can use this flag to mark which extension records should be copied.
Availability	Local, full, and lightweight clients.
Syntax	*toBeCopied* = *extension*.**CopyWithParent** **Element** — **Description** *toBeCopied* — A Boolean value indicating whether this extension record should be copied when its parent object is copied. **TRUE**. *Extension* object should be copied. **FALSE**. *Extension* object should not be copied.

	extension The *Extension* object to be acted upon.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Description

Summary	Stores description information about the *Extension* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *Extension* object. *object* The *Extension* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Extensions

The *Extensions* property can be used to retrieve a child *Extensions* collection from a parent *Extension* object. This property supports the concept of hierarchical *Extension* objects.

An *Extension* object cannot be removed from its parent object if the *Extension* has any child *Extension* objects. All child *Extension* objects must be removed first. An attempt to remove a non-empty *Extension* object will result in an **lcaErrCantRemoveExtensionWithChildren** (LCA #173) exception being thrown.

Summary	Retrieves a child *Extensions* collection from a parent *Extension* object.
Availability	Local, full, and lightweight clients.
Syntax	*extensions* = *extension*.*Extensions***()** **Element** **Description** *extension* The parent *Extension* object. *object* The child *Extensions* collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

Handle

Summary	Contains the handle associated with the *Extension* object. This property enables the *Extension* to be retrieved by the *ItemByHandle* method of the *Extensions* object.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *extension*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the *Extension* object. *object* — The *Extension* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	OpenLNS.

Key

Summary	Contains an application-specific unique ID used for keyed access to extension records. The key should be set to a unique value such as a globally unique ID (GUID) generated by many Windows development tools. Alternatively, the key may be a string starting with the developer's customer ID followed by a semicolon and an item identifier set by the developer. For example, the last program ID used may be identified with the key "9715A00-23145;LastProgramID". The customer ID is found on the inside cover of your LNS software CD case, and is always unique. You may not use the following characters in the *Key* property: forward slash (/), back slash (\), period (.), and colon (:).
Availability	Local, full, and lightweight clients.
Syntax	*keyValue* = *extensionObject*.**Key** **Element** — **Description** *extensionObject* — The *Extension* object to be operated on. *keyValue* — The key of the *Extension* as a string.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Owner

Summary	Returns the object that contains the *Extension* object. This property can be used to find the object that contains the

	Extensions collection containing this *Extension*. To determine the object type before using this method, use the *OwnerClassId* property.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *extension*.**Owner** **Element** **Description** *extension* The *Extension* object to be acted on. *object* The object that contains the *Extension*.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

OwnerClassId

Summary	Returns a constant identifying the type of object that contains the *Extension* object. This property can be used to determine the object type that contains the *Extensions* collection containing this *Extension*. To access the object that contains the *Extension*, use the *Owner* property. The possible values for this property are contained in the **ConstClassIds** constant.
Availability	Local, full, and lightweight clients.
Syntax	*classId* = *extension*.**OwnerClassId** **Element** **Description** *extension* The *Extension* object to be acted on. *classID* The type of object that "owns" the *Extension* object.
Data Type	Integer.
Read/Write	Read only.
Added to API	OpenLNS.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the

	specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Value1

Summary	Stores user-specified object data.
Availability	Local, full, and lightweight clients.
Syntax	*userData* = e*xtensionObject*.**Value1** **Element** — **Description** *userData* — The user-specific data. You can store data of any type supported by the Variant type in binary large objects (BLOBs) of up to 65,000 bytes. Values are tagged with a unique *key* to ensure that applications from multiple manufacturers do not overwrite each other's data. The *Value1* property supports the following Variant types: VT_BOOL VT_UI1 VT_I2 VT_I4 VT_CY VT_R4 VT_R8 VT_DATE VT_BSTR VT_ERROR VT_DISPATCH VT_UNKNOWN VT_EMPTY VT_NULL All single-dimensional Variant arrays, except for arrays of BSTRs. *extensionObject* — The *Extension* object to operate on.
Data Type	Variant.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Value2

Summary	Stores user-specified object data.
Availability	Local, full, and lightweight clients.
Syntax	*userData* = e*xtensionObject*.**Value2** **Element** — **Description** *userData* — The user-specific data. You can store data of any type supported by the Variant type in binary large objects (BLOBs) of up to 65,000 bytes. Values are tagged with a unique *key* to ensure that applications from multiple manufacturers do not overwrite each other's data. The *Value2* property supports the following Variant types: VT_BOOL VT_UI1 VT_I2 VT_I4 VT_CY VT_R4 VT_R8 VT_DATE VT_BSTR VT_ERROR VT_DISPATCH VT_UNKNOWN VT_EMPTY VT_NULL All single-dimensional Variant arrays, except for arrays of BSTRs. *extensionObject* — The *Extension* object to operate on.
Data Type	Variant.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Value3

Summary	Stores user-specified object data.
Availability	Local, full, and lightweight clients.
Syntax	*userData* = e*xtensionObject*.**Value3** **Element** — **Description** *userData* — The user-specific data. You can store data of any type supported by the

		Variant type in binary large objects (BLOBs) of up to 65,000 bytes. Values are tagged with a unique *key* to ensure that applications from multiple manufacturers do not overwrite each other's data. The *Value3* property supports the following Variant types: VT_BOOL VT_UI1 VT_I2 VT_I4 VT_CY VT_R4 VT_R8 VT_DATE VT_BSTR VT_ERROR VT_DISPATCH VT_UNKNOWN VT_EMPTY VT_NULL All single-dimensional Variant arrays, except for arrays of BSTRs.
	extensionObject	The *Extension* object to operate on.
Data Type	Variant.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

Extensions

The *Extensions* object is a collection of *Extension* objects. Each manufacturer can define any number of manufacturer specific data objects for application specific data. Collections of these objects can be associated with the object server, a channel, a device template, a hardware template, a network, a device, or a router.

You should note that the *Extensions* collection for a Network object is stored in the computer-specific global database, and is not exported with the Network database. As a result, the collection would be lost in the process of transferring a network database from one computer to another. The following table summarizes the *Extensions* object.

Description	Represents a collection of *Extensions* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object *Channel* object *DeviceTemplate* object *LonMarkObject* object *Network* object *NetworkServiceDevice* object

	ObjectServer object *Router* object *Subnet* object *Subsystem* object *System* object
Default Property	*Item* property.
Methods	• *Add* • *ItemByHandle* • *Remove*
Properties	• *ClassID* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *Extensions* object contains the following methods.

- *Add*
- *ItemByHandle*
- *Remove*

Add

Summary	Define a new *Extension* object.
Availability	Local, full, and lightweight clients.
Syntax	*extensionObject* = *extensionsColl*.**Add**(*extensionKey*) **Element** — **Description** *extensionObject* — The newly defined *Extension* object. *extensionsColl* — The *Extensions* collection object. *extensionKey* — A String containing the extension's *key* value.
Added to API	Prior to LNS Release 3.0.

ItemByHandle

You can use the *ItemByHandle* method to retrieve an *Extension* object from the *Extensions* collection by handle.

Summary	Retrieves an *Extension* object by its *handle* property.
Availability	Local, full, and lightweight clients.
Syntax	*extension* = *extensions*.**ItemByHandle** **Element** — **Description** *Extension* — The *Extension* object. *Extensions* — The *Extensions* collection.

	handle	The handle of the *Extension* to be retrieved.
Added to API	OpenLNS.	

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed. *name* — A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *Extensions* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Extensions* object in the *ConstClassIds* constant: **51 lcaClassIdExtensions** *object* — The object to be acted on.
Data Type	Integer.

Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an *Extension* object from an *Extensions* collection. You can retrieve an *Extension* object from its *Extensions* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *Extension* object retrieved from the collection. *collObject* — The *Extensions* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *Extension* object to be retrieved. *stringExpression* — A string type specifying the name of the *Extension* object to be retrieved.

Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.

Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

FileTransfer

A *FileTransfer* object represents a LonMark file transfer session involving a group of application devices. For example, you might set up a file transfer to upload log files from some of the application devices on your network.

You can identify the target application devices, which store the files that are to be read from or written to, for a file transfer using the *AddTarget* method.

You can set the file transfer parameters, such as the index number of the file to be written to or read from, using the properties of the *FileTransfer* object. Once you have set these properties, you can execute file transfers by invoking the *ReadFile* or *WriteFile* methods.

The following table summarizes the *FileTransfer* object.

Description	A LonMark file transfer session involving a group of application devices.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None.
Methods	• *AddTarget* • *ClearTargets* • *ReadFile* • *WriteFile*
Properties	• *AuthenticationFlag* • *ClassId* • *FileCount* • *FileIndex* • *FileInfo* • *FileSize* • *FileType* • *HostTimeOut* • *Parent* • *PriorityFlag* • *ReadBufferLength* • *RetryCount* • *RxTimeOut* • *StartPosition*

	• *TxTimeOut*

Methods

The *FileTransfer* object contains the following methods:

- *AddTarget*
- *ClearTargets*
- *ReadFile*
- *WriteFile*

AddTarget

Summary	Adds a target application device to the file transfer target list. Currently, you can perform file transfer implicitly without the use of a previously defined connection by the OpenLNS Server with only one device at a time. Once you have added an app device to the *FileTransfer* object's target list with this method, you can write files to the app device with the *WriteFile* method, or you can read from the files on the app device with the *ReadFile* method. You can also access information concerning the files stored on the device such as the number of files on the device, the size of each file on the device, and the type of each file on the device, by referencing the index number assigned to the app device in the target list and reading properties of the *FileTransfer* object such as the *FileCount*, *FileSize*, and *FileType* properties. The target devices are assigned index numbers in sequential order as they are added to the target list with this method, starting with index value 1. You can write to multiple files at a time, but doing so requires adding network variables to the OpenLNS Server's NSD, and connecting them to the target files. For more information on this, see the online help for the *WriteFile* method.
Availability	Local, full, and lightweight clients.
Syntax	*ftObject*.**AddTarget** *appDeviceObject* **Element** — **Description** *ftObject* — The *FileTransfer* object. *appDeviceObject* — The *AppDevice* object to add as a target.
Added to API	Prior to LNS Release 3.0.

ClearTargets

Summary	Clears the file transfer target list. After clearing all of the targets, the *ReadFile* and *WriteFile* methods will not function until at least one target is added.

Availability	Local, full, and lightweight clients.
Syntax	*ftObject*.**ClearTargets** **Element** **Description** *ftObject* The *FileTransfer* object.
Added to API	Prior to LNS Release 3.0.

ReadFile

Summary	Reads a file from an *AppDevice* in the file transfer target list. You must first add the selected application device to the file transfer target list with the *AddTarget* method. You can use this method to read from one device at a time. As a result, the file transfer target list must contain one only device when the *ReadFile* method is invoked. If it contains more than one device, an exception will be thrown. You can set the desired file transfer parameters (including file index of the file to be read on the app device, and the buffer read count) using the *FileIndex* and *ReadBufferLength* properties of the *FileTransfer* object.
Availability	Local, full, and lightweight clients.

Syntax	*fileBuffer = ftObject.ReadFile*
	Element **Description**
	fileBuffer A Variant containing the returned buffer.
	The returned value is a Variant containing a one dimensional byte array (MFC type VT_UI1 | VT_ARRAY).
	The number of elements in the array is equal to the *ReadBufferLength* property multiplied by the number of targets.
	In Visual Basic, the individual elements may be retrieved from the variable by appending an array index to the name of the Variant. For example:
	`Dim fileBuffer as Variant`
	`Dim byte1 as Byte`
	`fileBuffer = ftObject.ReadFile`
	`byte1 = fileBuffer(1)`
	ftObject The *FileTransfer* object.

Added to API	Prior to LNS Release 3.0.

WriteFile

Summary	Writes a file to an *AppDevice* in the file transfer target list. You must first add the selected application device to the file transfer target list with the *AddTarget* method of the *FileTransfer* object. The desired file transfer parameters (including the index number of the file to be written to on the app device) are set using the properties of the *FileTransfer* object. You can use this method to write a file buffer to more than one application device at a time. The file index written to, and the file buffer to write, must be the same for all target devices. In order to write to more than one application device at a time, an output network variable of type **SNVT_file_req** on the *NetworkServiceDevice* object of the OpenLNS Server (**System .NetworkServiceDevice.AppDevice .Interface.NetworkVariables**) must be bound to the file request input network variable on each of the file transfer target devices. In addition, an input network variable of type **SNVT_file_status** on the *NetworkServiceDevice* object of the OpenLNS Server must be bound to the file status output network variable on each of the target devices. If random access is used for the file transfer, then an input network variable of type **SNVT_file_pos** on the *NetworkServiceDevice* object of the OpenLNS Server must be bound to the file position input NV on each of the targets. Random access will be used in the file transfer if the *StartPosition* property of the *FileTransfer* object is set to a non-zero value. Please note that all of these connections must use group addressing.
Availability	Local, full, and lightweight clients.
Syntax	*ftObject.WriteFile fileBuffer* **Element** — **Description** *fileBuffer* — The file buffer may be specified as either or a Variant containing a byte array (MI VT_UI1 | VT_ARRAY). *ftObject* — The *FileTransfer* object.
Added to API	Prior to LNS Release 3.0.

Properties

The *FileTransfer* object contains the following properties:

- *AuthenticationFlag*
- *ClassId*
- *FileCount*
- *FileIndex*
- *FileInfo*

- *FileSize*
- *FileType*
- *HostTimeOut*
- *Parent*
- *PriorityFlag*
- *ReadBufferLength*
- *RetryCount*
- *RxTimeOut*
- *StartPosition*
- *TxTimeOut*

AuthenticationFlag

Summary	Specifies whether authenticated messages are used for the file transfer.
Availability	Local, full, and lightweight clients.
Syntax	*authFlag* = *fileTransObject*.**AuthenticationFlag** **Element** — **Description** *authFlag* — A Boolean value. **TRUE**. Authenticated messages are used to perform the file transfer. To perfrom an authenticated file transfer, all participating devices must have authentication enabled **FALSE**. Authenticated messages are not used to perform the file transfer. This is the default. *fileTransOject* — The *FileTransfer* object to be acted on.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description**

	classIdValue	The object class of the object. The following value is defined for the *FileTransfer* object in the *ConstClassIds* constant: **57 lcaClassIdFileTransfer**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

FileCount

Summary	Returns the number of files supported by any of the app devices that have been added to the *FileTransfer* object's target list. This property is only available when the device is connected and online.	
Availability	Local, full, and lightweight clients.	
Syntax	*countValue* = *fileTransObject*.*FileCount* (*nTarget)* **Element** / **Description** *countValue* — The file count. The count may include files implemented using the file transfer protocol, or files implemented via direct memory read/write. *fileTransObject* — The *FileTransfer* object to be acted on. *nTarget* — The *FileTransfer* object index. The *FileTransfer* object has a list of targets, numbered from one to N. *nTarget* is an index within that list, used to get the node handle of the app device for which the file count is to be obtained. If the *nTarget* value is out of range, an exception will be thrown.	
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

FileIndex

Summary	Specifies the file index to be used during operations on the *FileTransfer* object. **Note**: OpenLNS does not support writing to the template file or the value file (or value files) because it would corrupt the configuration properties. Instead, OpenLNS provides the *ConfigProperties* object to provide transparent access to configuration properties implemented via configuration network variables, file transfer, and direct memory read/write. OpenLNS applications must use the *ConfigProperties* objects in place of direct file transfer when writing or reading configuration properties.
Availability	Local, full, and lightweight clients.
Syntax	*indexValue* = *fileTransObject*.*FileIndex* **Element** — **Description** *indexValue* — The file index to read or write. The file index may have a value between 0–65,535. The default value is 2. • For a write, the value is the index on the receivers. • For a read, the value is the index on the sender. The file index property should be set before any of the *FileTransfer* object methods are invoked. LonMark compliant devices may implement standard configuration property types (SCPTs) using the LonMark file transfer protocol. In this case, file indexes **0** and **1** are reserved for the SCPT template and value files. Index **2** may be used for the constant value file. *fileTransObject* — The *FileTransfer* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

FileInfo

Summary	Returns a 16-byte string describing any of the files on the app devices that have been added to the *FileTransfer* object's target list. The *FileTransfer* object has a list of targets, numbered from one to *n*. The *nTarget* element is an index within that list, which is used to get the node handle of the app device for which a file description is to be obtained. If the *nTarget* value is out of range, an exception is thrown. The app device identified by the *nTarget* element may have multiple files. This property will return the file description of the file that is using the value assigned to the *FileIndex* property as its index number.
Availability	Local, full, and lightweight clients.
Syntax	*szFileInfoValue = fileTransObject*.**FileInfo** (*nTarget*) **Element** — **Description** *szFileInfoValue* — The file description. *fileTransObject* — The *FileTransfer* object to be acted on. *nTarget* — The *FileTransfer* object index
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

FileSize

Summary	Returns the file size of any of the files on the app devices that have been added to the *FileTransfer* object's target list, in bytes (up to 4 GB).
Availability	Local, full, and lightweight clients.
Syntax	*nFileSizeValue = ftObject.FileSize (nTarget)* **Element** — **Description** *nFileSizeValue* — The file size. *ftObject* — The *FileTransfer* object to be acted on. *nTarget* — The *FileTransfer* object index. The *FileTransfer* object has a list of targets, numbered from one to N. *nTarget* is an index within that list, used to get the node handle of the app device for which the file size is to be obtained. If the *nTarget* value is out of range, an exception will be thrown. The app device identified by the *nTarget* element may have multiple

	files. This property will return the file size of the file that is using the value assigned to the *FileIndex* property as its index number.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FileType

Summary	Returns the file type of any of the files on the app devices that have been added to the *FileTransfer* object's target list.
Availability	Local, full, and lightweight clients.
Syntax	*nFileTypeValue = ftObject.FileType (nTarget)* **Element** — **Description** *nFileTypeValue* — File type. *ftObject* — The *FileTransfer* object to be acted on. *nTarget* — The *FileTransfer* object index. The *FileTransfer* object has a list of targets, numbered from one to N. *nTarget* is an index within that list, used to get the node handle of the app device for which the file description is to be obtained. If the *nTarget* value is out of range, an exception will be thrown. The app device identified by the *nTarget* element may have multiple files. This property will return the file type of the file that is using the value assigned to the *FileIndex* property as its index number.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

HostTimeout

Summary	Specifies the host timeout value to be used during a file transfer
Availability	Local, full, and lightweight clients.

Syntax	*timeoutValue* = *fileTransObject*.**HostTimeOut** **Element** — **Description** *timeoutValue* — The timeout value. The time-out value in seconds (0–65,535). The number indicates the maximum number of seconds a file transfer setup or clearing request may take if the target is busy. The attempt will be made once per second. The default value is 30 seconds. *fileTransObject* — The *FileTransfer* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PriorityFlag

Summary	Specifies whether priority messaging is to be used during the file transfer.
Availability	Local, full, and lightweight clients.

Syntax	*priorityFlag* = *fileTransObject*.**PriorityFlag** **Element** — **Description** *priorityFlag* — The returned priority flag, which is a Boolean value. **TRUE**. Priority messaging is used. **FALSE**. Priority messaging is not used. *fileTransObject* — The *FileTransfer* object to be acted on.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ReadBufferLength

Summary	Specifies the read buffer length for file transfer. When reading data from a file on a device, this property allows the application to specify how many bytes to read, and how large to make the return value from the *ReadFile* method. For the *ReadFile* method, the size of the returned array, will be the *ReadBufferLength* multiplied by the number of target devices
Availability	Local, full, and lightweight clients.
Syntax	*bufferSize* = *fileTransObject*.*ReadBufferLength* **Element** — **Description** *bufferSize* — The read buffer length. *fileTransObject* — The *FileTransfer* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

RetryCount

Summary	Sets the maximum number of retries used by the sender during the request/response phase of the file transfer protocol.
Availability	Local, full, and lightweight clients.
Syntax	*retryCountValue* = *object*.**RetryCount** **Element** — **Description** *retryCountValue* — The retry count value. This value does not affect the retry count used by the initiator during setup.

		A value of **0** indicates that the number of retries will be calculated based on the network topology. The default value is **0**.
	object	The *FileTransfer* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

RxTimeOut

Summary	Specifies the receive timeout value used for file transfer messages.	
Availability	Local, full, and lightweight clients.	
Syntax	*timeoutValue* = *fileTransObject*.**RxTimeOut**	
	Element	**Description**
	timeoutValue	The returned receive timeout value in milliseconds (0–65,535). The file is automatically closed, and an exception will be thrown, if no file transfer messages are received during the specified interval. The default value is **0**, which means that the timeout is calculated based on the network topology.
	fileTransObject	The *FileTransfer* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

StartPosition

Summary	Specifies the position of the file pointer for a random access file operation.	
Availability	Local, full, and lightweight clients.	
Syntax	*positionValue* = *fileTransObject.StartPosition*	
	Element	**Description**
	positionValue	The start position value. A start position of **0** specifies sequential file access. A non-zero value causes a seek operation to be performed before executing the read or write.

		This method requires network communication; however, you can invoke it while the network management mode (*MgmtMode* property) is set to **lcaMgmtModeDeferConfigUpdates (1)**.
	fileTransObject	The *FileTransfer* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

TxTimeOut

Summary	Specifies the transmit timeout value used for file transfer messages. During a file transfer, request messages that require responses are sent as part of the file transfer protocol. The value assigned to this property represents the transmit time-out for this phase of the file transfer. The *RetryCount* property of the *FileTransfer* object determines how many times the request messages are attempted in case of failure, and the *TxTimeOut* property determines how much time is allocated to this phase of the file transfer. As a result, the values assigned to these properties determine how much time is allocated to each request message to be sent, and for each response message to be received. The time allocated to each request and response message can be calculated as follows: *transactionTime*= *fileTransObject*.**TxTimeOut**/(*fileTransferObject*.**RetryCount**+1) If you write a value other than the default to the *TxTimeOut* property, specify a value that will provide for an adequate request/response transaction time. Otherwise, your file transfer may fail.
Availability	Local, full, and lightweight clients.
Syntax	*timeoutValue* = *fileTransObject*.**TxTimeOut** **Element** — **Description** *timeoutValue* — The returned transmit timeout value in milliseconds (0–65,535). If this time-out period expires and no request messages have been sent, the file will be closed, and an exception will be thrown. The transmit timeout value must be

		less than or equal to the receive timeout value. The default value is 0,which means that OpenLNS calculates the timeout period based on the network topology and channel delays.
	fileTransObject	The *FileTransfer* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

FormatLocale

A *FormatLocale* object contains a series of properties that reflect a particular geographical area's conventions for data display. These conventions affect how data should be displayed in that area, including factors such as language, measurement system, date formats, time formats, and decimal number formats. The settings of a *FormatLocale* object determine how data stored in the *FormattedValue* properties of all *DataPoint* objects will be displayed when the *FormatLocale* object is used by an application.

The *FormatLocales* collection contains 4 pre-defined, read-only *FormatLocale* objects. For more information on the pre-defined *FormatLocale* objects, see the online help for the *FormatLocales* collection.

You can create custom *FormatLocale* objects with the *Add* method; however, your application can only use one *FormatLocale* object at a time. You can determine which *FormatLocale* object will be used by your application by writing to the *CurrentFormatLocale* property of the *ObjectServer* object.

Note: You can only write to the *CurrentFormatLocale* property, or modify the *FormatLocale* object acting as the *CurrentFormatLocale*, before your application has opened any networks and formatted any data. Operations that will cause your application to format data include acquiring a *DataPoint* object, and reading or writing the value of a *ConfigProperty* or *NetworkVariable* object. If you write to the *CurrentFormatLocale* property (or the *FormatLocale* object assigned to the property) after performing any of these operations, the **LCA, #122 lcaErrReadOnlyInContext** exception will be thrown.

In addition, the Object Server contains four pre-defined *FormatLocale* objects using index values 1, 2, 3 and 4. All attributes of these *FormatLocale* objects are read-only.

The following table summarizes the *FormatLocale* object.

Description	Contains a series of properties that reflect a particular geographical area's conventions for data display.
Added to API	LNS Release 3.20.
Accessed Through	*FormatLocales* collection object.
Default Property	None.
Methods	None.
Properties	• *CategoryPreferenceList* • *ClassId*

	• *DateFormatSeparator* • *DateFormatSeparatorSource* • *DecimalPointCharacter* • *DecimalPointCharacterSource* • *DoubleFloatPrecision* • *FallbackFormat* • *FloatPrecision* • *LanguageId* • *LanguageIdSource* • *ListSeparatorCharacter* • *ListSeparatorCharacterSource* • *MeasurementUnits* • *MeasurementUnitsSource* • *Name* • *Parent* • *ShortDateFormat* • *ShortDateFormatSource* • *ShortTimeFormat* • *ShortTimeFormatSource* • *TimeFormatSeparator* • *TimeFormatSeparatorSource*

Methods

The *FormatLocale* object does not contain any methods.

Properties

The *FormatLocale* object contains the following properties:

- *CategoryPreferenceList*
- *ClassId*
- *DateFormatSeparator*
- *DateFormatSeparatorSource*
- *DecimalPointCharacter*
- *DecimalPointCharacterSource*
- *DoubleFloatPrecision*
- *FallbackFormat*
- *FloatPrecision*
- *LanguageId*
- *LanguageIdSource*
- *ListSeparatorCharacter*
- *ListSeparatorCharacterSource*
- *MeasurementUnits*
- *MeasurementUnitsSource*
- *Name*
- *Parent*
- *ShortDateFormat*
- *ShortDateFormatSource*
- *ShortTimeFormat*
- *ShortTimeFormatSource*
- *TimeFormatSeparator*
- *TimeFormatSeparatorSource*

CategoryPreferenceList

Summary	Establishes the format to be used when displaying the formatted value of a data point whose default type contains several alternate formats, such as **SNVT_press**. OpenLNS will initially choose the format to use for data points like this based on the setting of the *MeasurementUnits* property. This allows you to choose between U.S. or SI metric measurement units. You can also use localized formats whenever possible by writing to the *CategoryPreferenceList* property. In this case, OpenLNS will first determine whether to use U.S. or SI metric formatting by reading the *MeasurementUnits* property. It will then use the *CategoryPreferenceList* property to determine if it should use the localized alternate formats that are available. If the *CategoryPreferenceList* property is not set, OpenLNS will use the first alternate format it finds that complies with the setting of the *MeasurementUnits* property. For example, consider a case where a data point is using the **SNVT_hvac_overid** type. If the *MeasurementUnits* property is set to use U.S. units, and the *CategoryPreferenceList* property is not set, OpenLNS will use the **SNVT_hvac_overid#US** format to display the value of the data point. However, if the *CategoryPreferenceList* property is set to "LO", OpenLNS will use the **SNVT_hvac_overid#US_LO** format to display the value. Consider another case where a data point is using the **SNVT_press type** and the *MeasurementUnits* property is set to use metric measurement units. If the *CategoryPreferenceList* property is not set, OpenLNS will use the **SNVT_press#SI** format to display the value of the data point. If the *CategoryPreferenceList* property is set to "kPa", OpenLNS will use the **SNVT_press#SI_kPa** format to display the value of the data point. This feature may also applies types that do not have separate U.S. or SI metric formats but still have localized alternate formats, such as **SNVT_alarm**. For example, if a data point is using the **SNVT_alarm** type and the *CategoryPreferenceList* is set to "LO", OpenLNS will use the **SNVT_alarm#LO** format to display the value of the data point. If the *CategoryPreferenceList* is not set to "LO", OpenLNS will use the **SNVT_alarm** format. In many cases, your application may need to display the values of data points that use different types. As a result, you can specify multiple alternate formats for a *FormatLocale* by passing them to the *CategoryPreferenceList* property in a comma-separated list. For example, you could set the *CategoryPreferenceList* to "LO,psi" to cause LNS to look for those alternate formats whenever possible. OpenLNS will

	search for matches for each alternate format specified starting with the first entry in the list. If you specify the same format more than once in the list you write to the *CategoryPreferenceList* property, the **Formatter, #40 lcaErrFormatStringDuplicate** exception will be thrown. You should be aware that the *CategoryPreferenceList* property has no effect on data points acquired through network variables or configuration properties whose format has been previously modified by an OpenLNS application (for example, by writing to the *DsFormatType* property of the source *NetworkVariable* object, or by writing to the *FormatName* property of the source *ConfigProperty* object).
Availability	Local, full, and lightweight clients.
Syntax	*formatLocale.CategoryPreferenceList* = *preference* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *preference* — Set this property to cause OpenLNS to use localized formats when displaying formatted values whenever possible; otherwise, do not write to the property.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *FormatLocale* object in the *ConstClassIds* constant: **95 lcaClassIdFormatLocale** *object* — The object to be acted on.

Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

DateFormatSeparator

Summary	Determines which symbol is used to separate the digits that represent months, days and years when the formatted value is displayed as a date. For example, if this property is set to the backslash (/) character, dates would be displayed as follows: 11/26/1977, 8/13/83, etc. You can determine the format and order used to display the month, day and year digits with the *ShortDateFormat* property. This applies to format specifications containing the `date()` macro in their text format specification, such as SNVT_data_cal#LO: `text(date(year, month, day))` You can use the *DateFormatSeparatorSource* property to determine how the *DataFormatSeparator* property should be filled in.
Availability	Local, full, and lightweight clients.
Syntax	*formatLocale*.**DateFormatSeparator =** *character* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *character* — The symbol chosen to to separate digits representing months, days and years when a formatted value is displayed as a date. You can choose any printable ASCII character.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

DateFormatSeparatorSource

Summary	Determines whether OpenLNS should assign the *DataFormatSeparator* property a value automatically based on the Windows control panel Regional Options settings.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale.DateFormatSeparatorSource* = *source* **Element** — **Description**

	formatLocale	The *FormatLocale* object being acted upon.
	source	The value of the *DateFormatSeparatorSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

DecimalPointCharacter

Summary	Determines which will be used to indicate decimal places when a formatted value is displayed as a scalar number. For example, if this property is set to the period character, scalar values with decimal places would be displayed as follows: 100.34, 78.4, 99.6, and so on. This applies to format specifications that use the `%f` symbol in their text format specification, such as **SNVT_temp#US**: `text("%f", *1.8+32(0:855))` You can use the *DecimalPointCharacterSource* property to determine how the *DecimalPointCharacter* property should be filled in.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*.**DecimalPointCharacter =** *character* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *character* — The specified symbol used to indicate decimal places for this *FormatLocale* object. You can choose any printable ASCII character.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

DecimalPointCharacterSource

Summary	Determines whether OpenOpenLNS should assign the *DecimalPointCharacter* property a value automatically based on the Windows control panel Regional Options settings.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale.DecimalPointCharacterSource* **=** *source* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *source* — The value of the *DecimalPointCharacterSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource**

	UserDefaultRegionalSetting Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

DoubleFloatPrecision

Summary	Determines the precision that will be used when displaying double-float values. This property is implemented when the data stored in the *FormattedValue* property is formatted as an ASCII representation of a double precision floating-point value. It specifies the default number of digits that will be used to display the value. **Note**: The *FormatSpec* object of each data point contains a *Precision* property. This sets the precision that will be used to display data stored in that data point. The *DoubleFloatPrecision* property will only be used for a data point if its *Precision* property is not set.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*.**DoubleFloatPrecision =** *precision* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *precision* — The double-float precision to use when displaying data with this *FormatLocale* object. The element has a range between 0–17. This default value is **15**. You can write **–1** to this property at any time to return it to its default value.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

FallbackFormat

Summary	Specifies the default type that should be used to display a data point's formatted value if the data point's actual type cannot be determined. This may be necessary if a data point's type is unknown, or if it cannot be found in the resource files. Typically, OpenLNS will automatically assign each data point a type based on the type of the network variable or configuration property associated with the data point. You can also manually set a data point's type by writing to the *FormatType* property of the data point's *FormatSpec* object. This property only applies to data points accessed through network variables.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*.**FallbackFormat =** *type* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *type* — The fallback type selected for the *FormatLocale* object. The valid settings for this property, which are contained in the *ConstFallbackFormatTypes* constant, are as follows: **0 lcaFormatRaw** Use "RAW" as the fallback type. This is a text string format, and is the default format for user-defined network variables (UNVTs). Each byte of the network variable value (in big-endian format) appears as a text-formatted integer value from "0" through "255". Each value byte is separated by the TAB character (0x09). **1 lcaFormatRawHex** Use "RAW_HEX" as the fallback type. This is a text string format. Each byte of the network variable value (in big-endian format) appears as a text-formatted hex integer value from "0" through "ff". Each value byte is separated by the

		TAB character (0x09). **2 lcaFormatRawHexPacked** Use "RAW_HEX_PACKED" as the fallback type. This is a text string format, just like the "RAW_HEX" format. Each byte of the network variable value (in big-endian format) appears as a text-formatted hex integer value from "0" through "ff". The value bytes are not separated by the TAB character when this type is used.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

FloatPrecision

Summary	Determines the precision that will be used when displaying single-float values. This property is implemented when the data stored in the *FormattedValue* property is formatted as an ASCII representation of a single precision floating-point value. It specifies the default number of digits that will be used to display the value. **Note**: The *FormatSpec* object of each data point contains a *Precision* property. This sets the precision that will be used to display data stored in that data point. The *FloatPrecision* property will only be used for a data point if its *Precision* property is not set.	
Availability	Local, full, lightweight, and independent clients.	
Syntax	*formatLocale*. **FloatPrecision =** *precision*	
	Element	**Description**
	formatLocale	The *FormatLocale* object being acted upon.
	precision	The single-float precision to use when displaying data with this *FormatLocale* object. The element has a range between 0–7. This default value is **6**. You can write **–1** to this property at any time to return it to its default value.

Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

LanguageId

Summary	Determines the language that will be used to display Windows localized settings that are stored in the formatted values. For example, if the formatted value contains a date, this would determine the language used to display the name of the month in the date. All language codes are three characters long. For example, you could enter "ENU" to display the information in U.S. English, "FRA" for French, or "DEU" for German. You can use the *LanguageIdSource* property to determine how the *LanguageId* property should be filled in.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*.**LanguageId =** *code* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon.

	code	A 3-letter identifier for the language specified for the *FormatLocale* object. The available language codes are as follows: Czech CSY Danish DAN Dutch (Belgian) NLB Dutch (default) NLD English (UK) ENG English (US) ENU Finnish FIN French (Belgian) FRB French (Canadian) FRC French (default) FRA French (Swiss) FRS German (Austrian) DEA German (default) DEU German (Swiss) DES Greek ELL Hungarian HUN Icelandic ISL Italian (default) ITA Italian (Swiss) ITS Norwegian (Bokmal) NOR Polish PLK Portuguese (Brazilian) PTB Portuguese (default) PTG Russian RUS Slovak SKY Spanish (default) ESP Spanish (Mexican) ESM Swedish SVE
	ldrfObject	The *LdrfLanguage* object to be acted on.
Data Type	String.	
Read/Write	Read only.	
Added to API	LNS Release 3.20	

LanguageIdSource

Summary	Determines whether OpenLNS should assign the *LanguageId* property a value automatically based on the Windows control panel Regional Options settings
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*.**LanguageIdSource =** *source* **Element** **Description**

	formatLocale	The *FormatLocale* object being acted upon.
	source	The value of the *LanguageIdSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

ListSeparatorCharacter

Summary	Determines which symbol is used to separate items in the formatted value that are returned as parts of a list. This applies to format specifications that use the bar character ("|") to represent a localized list separator character. A few examples of this are **SNVT_hvac_status#LO** and **SNVT_magcard#LO**. You can use the *ListSeparatorCharacterSource* property to determine how the *ListSeparatorCharacter* property should be filled in.
Availability	Local, full, and lightweight clients.
Syntax	*formatLocale*.**ListSeparatorCharacter =** *character* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *character* — The symbol chosen as the list separator character. You can choose any printable ASCII character.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

ListSeparatorCharacterSource

Summary	Determines whether OpenLNS should assign the *ListSeparatorCharacter* property a value automatically based on the Windows control panel Regional Options settings
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*.**ListSeparatorCharacterSource =** *source* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *source* — The value of the *ListSeparatorCharacterSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically,

		based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

MeasurementUnits

Summary	Determines the measurement units (Systeme Internationale [SI metric] or U.S.) that will be used to display the formatted values of data points when this *FormatLocale* object is used by an application. This is implemented when a data point is using a default type that has several alternate formats associated with it. For example, consider a case where a data point is using **SNVT_temp** as its default type. The formatted value of the data point would be displayed using the **SNVT_temp#US** format if this property is set to **lcaMeasurementUnitsUS (1)**, or using the **SNVT_temp#SI** format if this property is set **lcaMeasurementUnitsSI (0)**. You can also cause LNS to choose localized formats whenever possible by writing the value "LO" to the *CategoryPreferenceList* property. In this case, OpenLNS will first determine whether to use U.S. or metric measurement units by reading the *MeasurementUnits* property. If the *CategoryPreferenceList* is set to "LO", and a localized alternate type that complies with the measurement units selected exists for the format used by the data point, OpenLNS will use the localized type. Otherwise, it will use the first alternate type it finds that complies with the setting of the *MeasurementUnits* property. For example, consider a case where a data point is using the **SNVT_hvac_overid** type. If the *MeasurementUnits* property is set to use U.S. units, and the *CategoryPreferenceList* property is not set, LNS would use the **SNVT_hvac_overid#US** format to display the value of the data point, as described earlier in this section. However, if the *CategoryPreferenceList* property is set to "LO", LNS would use the **SNVT_hvac_overid#US_LO** format to display the value. This may be useful if you want your application to use localized formats as much as possible. You can use the *MeasurementUnitsSource* property to determine how the *MeasurementUnits* property should be filled in.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale.MeasurementUnits* **=** *units* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *units* — The measurement units being used. The possible values for this element, which are stored in the *ConstFormatLocaleMeasurementUnits* constant, are as follows:

		0 lcaMeasurementUnitsSI Displays the formatted values using the metric measurement system. **1 lcaMeasurementUnitsUS** Displays the formatted values using the U.S. measurement system.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

MeasurementUnitsSource

Summary	Determines whether OpenLNS should assign the *MeasurementUnits* property a value automatically based on the Windows control panel Regional Options settings.	
Availability	Local, full, lightweight, and independent clients.	
Syntax	*formatLocale*.**MeasurementUnitsSource =** *source*	
	Element	**Description**
	formatLocale	The *FormatLocale* object being acted upon.
	source	The value of the *MeasurementUnitsSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer

	running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent

	clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ShortDateFormat

Summary	Determines how a formatted value will be displayed if it represents a date. You can display the month, day and year in any order, but there are certain restrictions to how each part of the date can be displayed. This applies to format specifications containing the `date()` macro in their text format specification, such as **SNVT_date_time#LO**: `text(time(hour, minute, second))` • When setting the year format, you can the following formats: o "yy" to display the year with 2 digits. o "yyyy" to display the year with 4 digits. • When setting the month format, you can the following formats: o "M" to display the month with the least possible number of digits. o "MM" to always display the month with 2 digits, with leading zeroes for months 1-9. o "MMM" to display the month as a 3-letter, abbreviated name (e.g. "JAN" for January). o "MMMM" to display the month by its full name, as a string. • When setting the day format, you can use the following formats: o "dd" to always display the day with 2 digits, with leading zeroes for days 1-9. o "d" to display the day with the least possible nimber of digits.

	Here are some example strings you could use for this property, with examples of how the short date would appear for each one. The following examples assume the *DateFormatSeparator* property is set to /. *ShortDateFormat* **Example** MM/dd/yy 08/26/03 d/M/yyyy 26/8/2003 MMM/dd/yyyy Aug/26/2003 yy/dd/MMMM 03/26/August If you specify a character other than M, d, y, or the *DateFormatSeparator* character when writing to this property, the **LCA, #87 lcaErrStringInvalidChar** exception will be thrown. If you change the *DateFormatSeparator* property after setting the *ShortDateFormat* property, OpenLNS will automatically update it with the new separator character for you. You can use the *ShortDateFormatSource* property to determine how the *ShortDateFormat* property should be filled in.
Availability	Local, full, and lightweight clients.
Syntax	*formatLocale.ShortDateFormat* **=** *dateFormat* **Element** **Description** *formatLocale* The *FormatLocale* object being acted upon. *dateFormat* The format used to display dates. The allowable characters are M, d, y, and the character specified as the *DateFormatSeparator* property.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

ShortDateFormatSource

Summary	Determines whether OpenLNS should assign the *ShortDateFormat* property a value automatically based on the Windows control panel Regional Options settings.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale.* **ShortDateFormatSource =** *source* **Element** **Description** *formatLocale* The *FormatLocale* object being acted upon.

	source	The value of the *ShortDateFormatSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

ShortTimeFormat

Summary	Determines how the formatted value of a data point will be displayed if it represents a time. You can display the hours, minutes and seconds in any order, but there are certain

	restrictions to how each part of the time can be displayed. This applies to format specifications containing the `time()` macro in their text format specification, such as **SNVT_Time_time#LO**: `text(time(hour, minute, second))` • When setting the format used to display hours, you can use the following formats: o "H" to display the hours with as few digits as possible. o "HH" to display the always hours with double digits, with leading zeros for hours 0-9. Note that OpenLNS will display all times in 24-hour format. • When setting the format used to display minutes, you can use the following formats: o "m" to display the minutes with as few digits as possible. o "mm" to always display the minutes with double digits, with leading zeros for minutes 0-9. • When setting the format used to display seconds, you can use the following formats: o "s" to display the seconds with as few digits as possible. o "ss" to always display the seconds with double digits, with leading zeros for seconds 0-9. Note that time formats that include milliseconds will append the three-digit milliseconds to the seconds field, separated by a decimal point character. Here are some example strings you could use for this property, with examples of how a short time would appear for each one. **ShortTimeFormat** **Example** H/m/s 3:41:7.111 HH/m/ss 03:41:07.111 H/mm/ss 15:06:33.333 The following examples assume the *TimeFormatSeparator* property is set to the colon character (:). If you specify a character other than M, d, y, or the *TimeFormatSeparator* character when writing to this property, the **LCA, #87 lcaErrStringInvalidChar** exception will be thrown. If you specify additional characters that would cause LNS to display times in 12-hour format, the **LCA, #160 lcaErr12HourTimeFormatNotSupported** exception will be thrown. If you change the *TimeFormatSeparator* property after setting the *ShortTimeFormat* property, OpenLNS will

	automatically update it with the new separator character for you. You can use the *ShortTimeFormatSource* property to determine how the *ShortTimeFormat* property should be filled in.
Availability	Local, full, and lightweight clients.
Syntax	*formatLocale*.**ShortTimeFormat =** *timeFormat* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *timeFormat* — The format used to display dates. The allowable characters are "H", "m", "s" and the character specified as the *TimeFormatSeparator* property.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

ShortTimeFormatSource

Summary	Determines whether OpenLNS should assign the *ShortTimeFormat* property a value automatically based on the Windows control panel Regional Options settings.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*. **ShortTimeFormatSource =** *source* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *source* — The value of the *ShortTimeFormatSource* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows

	control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

TimeFormatSeparator

Summary	Determines which symbol will be used to separate digits representing hours, minutes and seconds when the formatted value is displayed as a time. For example, if this property is set to the colon (:) character, times would be displayed as follows: 1:12:33, 11:22:55, and so on. You can determine the format and order used to display the hour, minute and second digits with the *ShortTimeFormat* property. This applies to format specifications containing the `time()` macro in their text format specification, such as SNVT_date_time#LO: `text(time(hour, minute, second))` You can use the *TimeFormatSeparatorSource* property to determine how the *TimeFormatSeparator* property should be filled in.
Availability	Local, full, and lightweight clients.

Syntax	*formatLocale*.**ShortTimeFormat =** *timeFormat* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *timeFormat* — The format used to display dates. The allowable characters are "H", "m", "s" and the character specified as the *TimeFormatSeparator* property.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

TimeFormatSeparatorSource

Summary	Determines whether OpenLNS should assign the *TimeFormatSeparator* property a value automatically based on the Windows control panel Regional Options settings.
Availability	Local, full, lightweight, and independent clients.
Syntax	*formatLocale*. **TimeFormatSeparatorSource =** *source* **Element** — **Description** *formatLocale* — The *FormatLocale* object being acted upon. *source* — The value of the *TimeFormatSeparator* property. The possible values for this element, which are contained in the *ConstFormatLocaleSource* constant, are as follows: **0 lcaFormatLocaleSource UserDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically, based on the Windows regional settings set by the user on the computer running your application. A user can change the regional settings on a computer from the system defaults using the Windows control panel Regional Options applet. This is the default. **1 lcaFormatLocaleSource SystemDefaultRegionalSetting** Select this value to have OpenLNS set the associated property automatically,

		based on the system default Windows regional settings of the computer running your application. The default settings may vary depending on which operating system is installed on the computer running the application. Consult the *Microsoft Developer's Network* documentation for more information on this. **2 lcaFormatLocaleSource ManualSetting** This value indicates that the value of the associated property has been set manually by your application.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

FormatLocales

The *FormatLocales* property contains all the *FormatLocale* objects that have been added to the Object Server. It contains four pre-defined *FormatLocale* objects, which are described below. The pre-defined *FormatLocale* objects are read-only, but you can create custom *FormatLocale* objects with the *Add* method.

To retrieve a *FormatLocale* object from this collection, use the *Item* property and specify the object to be retrieved by its index number. The index number will be in the range of 1 to *x*, where *x* represents the value stored in the *Count* property.

Note that indexes 1, 2, 3 and 4 are assigned to the pre-defined, read-only *FormatLocale* objects. The pre-defined *FormatLocale* objects are listed by index and name, and summarized as follows:

- **Index 1, UserDefaultRegionalSettings.** This is the default value for the *CurrentFormatLocale* property. When you use this *FormatLocale* object as the *CurrentFormatLocale*, all the properties will be set based on the user-defined Windows regional settings on the computer running your application. You can change the regional settings on a computer using the Windows control panel Regional Options applet. Consult the Microsoft Developer's Network documentation of the Win32 `GetLocaleInfo()` function for more information on this.
- **Index 2, SystemDefaultRegionalSettings.** When you use this *FormatLocale* object as the *CurrentFormatLocale*, all the properties will be set based on the default Windows regional settings on the computer running the application. The default settings may vary, depending on which operating system is installed on the computer. Consult the Microsoft Developer's Network documentation of the Win32 `GetLocaleInfo()` function for more information on this.
- **Index 3, LonMarkCompatibility.** When you use this *FormatLocale* object as the *CurrentFormatLocale*, all properties will be set so that all formatted data will be displayed per LonMark standards used prior to LNS 3.0, when localized formatting was

not available. In this case, Systeme Internationale measurement units, and U.S. options for everything else, will be used to display all formatted data.

- **Index 4, ISO8601DateAndTime.** When you use this *FormatLocale* object as the *CurrentFormatLocale*, all properties will be set to be the same as the `LonMark Compatibility` settings, except for the localized time and date formats, which will be based on the ISO 8601 standard. This standard helps avoid confusion that may be caused by the many different national notations used for dates and times, and increases the portability of computer user interfaces.

  For more details on these *FormatLocale* objects, including the settings used for each property, see the *OpenLNS Programmer's Guide*.

You can determine which *FormatLocale* object will be used by your application by writing to the *CurrentFormatLocale* property of the *ObjectServer* object. Note that you can only write to the *CurrentFormatLocale* property and the *FormatLocale* object acting as the *CurrentFormatLocale* before your application has opened any networks and formatted any data. Operations that will cause your application to format data include acquiring a *DataPoint* object, and reading or writing the value of a *ConfigProperty* or *NetworkVariable* object. If you write to the *CurrentFormatLocale* property (or to the *FormatLocale* object assigned to the *CurrentFormatLocale* property) after performing any of these operations, the **LCA, #122 lcaErrReadOnlyInContext** exception will be thrown.

The following table summarizes the *FormatLocales* object.

Description	Contains all the *FormatLocale* objects that have been added to the Object Server.
Added to API	LNS Release 3.20.
Accessed Through	*ObjectServer* object.
Default Property	*Item*.
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *FormatLocales* contains the following methods.

- *Add*
- *Remove*

Add

Summary	Defines a new *FormatLocale* object. A *FormatLocale* object contains a series of properties that reflect a geographical area's conventions. These properties determine how data stored in the *FormattedValue* properties of *DataPoint* objects will be displayed when that *FormatLocale* is used by an application. This includes options

	such as the format used to display dates and times, and the unit of measurement associated with the formatted value. The default values assigned to the properties of the *FormatLocale* object match the settings of the pre-defined "UserDefaultRegionalSettings" *FormatLocale* object. This means that the default will be set based on the user-defined Windows regional settings on the computer running the application. For more information on these settings, see the *OpenLNS Programmer's Guide*. After you have created a *FormatLocale* object, you should set its properties to meet your application's requirements. Then, you can make it the active *FormatLocale* object on the system by writing to the *CurrentFormatLocale* property of the *ObjectServer* object. **Note:** You can only write to the *CurrentFormatLocale* property or to the *FormatLocale* object acting as the *CurrentFormatLocale* before your application has opened any networks and formatted any data. Operations that will cause your application to format data include acquiring a *DataPoint* object, and reading or writing the value of a *ConfigProperty* or *NetworkVariable* object. If you write to the *CurrentFormatLocale* property (or if you write to the *FormatLocale* object assigned to the *CurrentFormatLocale* property) after performing any of these operations, the **LCA, #122 lcaErrReadOnlyInContext** exception will be thrown
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*newFormatLocale* = *flCollection*.**Add** *name* **Element** — **Description** *newFormatLocale* — The *FormatLocale* object returned by the function. This is the newly defined *FormatLocale* object. *flCollection* — The *FormatLocales* collection being acted upon. *name* — The name of the new *FormatLocale* object. This can be a maximum of 85 characters long. The forward slash (/), back slash (\), period (.), and colon (:) characters may not be used in the name.
Added to API	LNS Release 3.20.

Remove

Summary	Removes a *FormatLocale* object from the collection. The *indexName* element is a Variant type, which allows you to specify the *FormatLocale* to be deleted by either its name, or by the index number assigned to it within the

	FormatLocales collection. You can determine the name of a *FormatLocale* object by reading its *Name* property. Note that the *FormatLocales* collection contains 4 pre-defined *FormatLocale* objects, using indices 1–4, that cannot be deleted. If you attempt to delete these *FormatLocale* objects with this method, the **LCA, #122 lcaErrReadOnlyInContext** exception will be thrown. If you delete the *FormatLocale* object acting as the *CurrentFormatLocale*, your application will use the **UserDefaultRegionalSettings** *FormatLocale* object (index 1) until you write a new value to the *CurrentFormatLocale* property. You can only use this method to delete the *FormatLocale* object acting as the *CurrentFormatLocale* (and you can only write to the *CurentFormatLocale* property) before your application has opened any networks or formatted any data. Operations that will cause your application to format data include acquiring a *DataPoint* object, and reading or writing the value of a *ConfigProperty* or *NetworkVariable* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*flCollection*.**Remove** *indexName* **Element** — **Description** *flCollection* — The *FormatLocales* collection being acted upon. *indexName* — A Long value specifying the collection index of the *FormatLocale* object to remove, or a String value specifying the name of the *FormatLocale* object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *FormatLocales* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *FormatLocales* object in the *ConstClassIds* constant: **96 lcaClassIdFormatLocales** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *FormatLocale* object from a *FormatLocales* collection. You can retrieve a *FormatLocale* object from its

	FormatLocales collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *FormatLocale* object in *FormatLocales* collections that contain objects with the *Name* property by passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *FormatLocale* object retrieved from the collection. *collObject* — The *FormatLocales* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *FormatLocale* object to be retrieved. *stringExpression* — A string type specifying the name of the *FormatLocale* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.

Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

FormatSpec

A *FormatSpec* object specifies the format to use when reading the formatted value a data point or monitor point. The information in this object is used to lookup information OpenLNS uses to format the values in the device resource catalog, such as unit conversion factors and precision settings.

The *FormatSpec* object contained within the *FormatSpec* property of each *DataPoint* and *NvMonitorPoint* object (as well as within the *InputFormatSpec* and *OutputFormatSpec* properties of each *MsgMonitorPoint* object) is not passed by reference. If you modify the values assigned to the properties of a local *FormatSpec* object, you must then explicitly assign the modified *FormatSpec* object back to the *FormatSpec* property of the *NvMonitorPoint* or *DataPoint* object (or to the *InputFormatSpec* and *OutputFormatSpec* properties of each *MsgMonitorPoint* object) for the changes to take effect. This following code sample demonstrates this procedure:

```
Set fsObject = dpObject.FormatSpec
fsObject.FormatName = "SNVT_temp_f#SI"
Set dpObject.FormatSpec = fsObject
```

After the *FormatSpec* object has been modified and passed back to the data point or monitor point, you can examine it to determine whether or not OpenLNS has found the specified format in the resource files. Because of changes to the behavior of the *FormatSpec* object since LNS Release 3.0, and because further changes are anticipated, you should use the *AltFormatNamesCount* property of the *FormatSpec* object to make this determination.

- If the *AltFormatNamesCount* property is set to a value greater than 0, it means that the specified format (as well as any alternate formats defined in the resource files) has been found, and is currently assigned to the data point or monitor point.
- If the *AltFormatNamesCount* property is set to 0, it means that type and format specification information for the referenced format was not found, and a built-in format with no associated type information has been assigned to the data point or monitor point. If this is the case, OpenLNS will use a fallback format, as determined by the *FallbackFormat* property of the *FormatLocale* object the application is using.

The following table summarizes the *FormatSpec* object.

Description	The format to be used when reading the formatted value a data point or monitor point.
Added to API	LNS Release 3.0.
Accessed Through	*DataPoint* object. *MsgMonitorPoint* object. *NvMonitorPoint* object.
Default Property	None.
Methods	None.
Properties	• *AltFormatName* • *AltFormatNamesCount* • *ClassId* • *FormatName* • *FormatType* • *Index* • *Parent* • *Precision* • *ProgramId* • *Scope* • *Units* • *UnitsAdder* • *UnitsMultiplier*

Methods

The *FormatSpec* object does not contain any methods.

Properties

The *FormatSpec* object contains the following properties:

- *AltFormatName*
- *AltFormatNamesCount*

- *ClassId*
- *FormatName*
- *FormatType*
- *Index*
- *Parent*
- *Precision*
- *ProgramId*
- *Scope*
- *Units*
- *UnitsAdder*
- *UnitsMultiplier*

AltFormatName

Summary	Contains an indexed list of all formats that can be applied to this data type. This allows you to search for other *FormatSpec* objects that contain alternate formats. For example, in a *FormatSpec* object representing the **SNVT_temp_f#US format** (for example, degrees Fahrenheit), this property would contain "**SNVT_temp_f#US** ", "**SNVT_temp_f#SI**", and "**SNVT_temp_f#US_diff**". The *AltFormatNamesCount* property indicates how many alternate formats are available.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*altName= fsObject.AltFormatName index* **Element** — **Description** *altName* — The alternate format. *fsObject* — The *FormatSpec* object to be acted on. *index* — If multiple formats for this type exist, the index number of the format to be returned. The index number must be between 1 to *x*, where *x* represents the number of alternate formats available.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

AltFormatNamesCount

Summary	Indicates how many alternate formats are contained in the *AltFormatName* property.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*altNameCount = fsObject.AltFormatNamesCount* **Element** — **Description** *altNameCount* — The number of alternate formats contained in the *AltFormatName* property. *fsObject* — The *FormatSpec* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue = object.***ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *FormatSpec* object in the *ConstClassIds* constant: **73 lcaClassIdFormatSpec** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

FormatName

Summary	Sets the name of the type to use when displaying the *FormattedValue* property of the *DataPoint* or *NvMonitorPointDataPoint* object using this *FormatSpec*. If the *FormatSpec* object's *FormatType* property is set to **lcaFormatTypeNv (1)**, OpenLNS will use the *Index* property to determine the type associated with the *FormatSpec* object. In this case, the *FormatName* property has no effect. The *FormatName* property will only be used if the

	FormatType property is set to **lcaFormatTypeNamed (0)**. In that case, the *Scope* and *ProgramId* properties are used to identify the resource file containing the type's definition. The way that OpenLNS determines the default value for this property, and the syntax you can use when writing to this property, varies depending on the type of object to which the *FormatSpec* applies: • *NvMonitorPoint* object or a *DataPoint* object that represents a network variable using a standard type. The *FormatType* property will be initially set to **lcaFormatTypeNv (1)**, and this property will have no effect. However, you can still set the *FormatType* property to **lcaFormatTypeNamed (0)** and then write to the *FormatName* property using the same syntaxes you would use when setting a *NetworkVariable* object's *DsFormatType* property. • *DataPoint* object that represents a network variable using a user-defined type. The *FormatType* property will be initially set to **lcaFormatTypeNamed (0),** and the *FormatName* property will default to the *TypeName* assigned to the *DataPoint* object. In this case, you can also write to the *FormatName* property using the same syntaxes you would use to write to a *NetworkVariable* object's *DsFormatType* property. See the *DsFormatType* property for more information. • *DataPoint* object that represents a configuration property. The *FormatType* property will be initially set to **lcaFormatTypeNamed (0),** and the *FormatName* property will default to the *TypeName* assigned to the *DataPoint* object. You can write to the *FormatName* property using the same syntaxes you would use to write to a *ConfigProperty* object's *FormatName* property. OpenLNS will determine the default value for the property in the same manner, as well. Consult the online help for the *FormatName* property of the *ConfigProperty* object for more information on this. **Note**: *FormatSpec* objects are not passed by reference. If you modify the values assigned to the properties of a *FormatSpec* object, you must then explicitly assign the *FormatSpec* object back to the *FormatSpec* property of the *NvMonitorPoint* or *DataPoint* for the changes to take effect.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*formNameValue = fsObject.FormatName*

	Element	**Description**
	fsObject	The *FormatSpec* object to be acted on.
	formNameValue	The *FormatName* as a string.
Data Type	String.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

FormatType

Summary	Determines the base type of the data read from and written to *FormattedValue* property in the *DataPoint* or *NvMonitorPoint* object using the *FormatSpec* object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*formatType = fsObject.FormatType* **Element** — **Description** *formatType* — The type for the data. Typically, OpenLNS will automatically assign each data point a type based on the type of the network variable or configuration property associated with the data point. You can use this property to override that setting. Also, you can set options to use when displaying the formatted value of data point objects by writing to the *CurrentFormatLocale* being used by your application. The enumerated values for this element, which are contained in the *ConstLNSFormatType* constant, are as follows: **0 lcaFormatTypeNamed** The *FormatSpec* object contains a format for a user defined network variable or configuration property type. If the type is a user-defined network variable or configuration property type (UNVT or UCPT), set this property to **lcaFormatTypeNamed(0)**, set the *FormatName* property to the name of the user type, and set the *Scope* and *ProgramId* properties to the scope and Program ID of the format file containing the definition of the type.

		The **lcaFormatTypeNamed(0)** value is the default value for user-defined network variables. **1 lcaFormatTypeNv** The *FormatSpec* object contains a format for a standard network variable type. If the type is a Standard Network Variable Type (SNVT), set this property to **lcaFormatTypeNv (1)**, and set the Index property to the SNVT index. The **lcaFormatTypeNv (1)** value is the default value for data points and monitor points acquired through standard network variables. **2 lcaFormatTypeCp** The *FormatSpec* object contains a format for a standard configuration property type. If the type is a Standard Configuration Property Type (SCPT), set this property to **lcaFormatTypeCp (2)**, and set the *Index* property to the SCPT index. The **lcaFormatTypeCp (2)** value is the default value for data points acquired through configuration properties.
	fsObject	The *FormatSpec* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

Index

Summary	Contains the SNVT or SCPT index of the type. • If the *FormatType* property is set to **lcaFormatTypeCp (2)**, this property contains the SCPTindex. • If the *FormatType* property is set to **lcaFormatTypeNv (1)**, this property contains the SNVTindex. • If the *FormatType* property is set to **lcaFormatTypeNamed (0)**, this property is not used.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.

Syntax	*indexValue* = *fsObject*.**Index** **Element** **Description** *fsObject* The *FormatSpec* object to be acted on. *indexValue* The index of the SNVT or SCPT.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Precision

Summary	Determines the precision of the *FormatSpec* object. This represents the number of digits that will be used when data is read and displayed using the format specification. This information is only used if the data being displayed is a floating-point type. If the *FormatSpec* object represents a single float type, this property defaults to the value of the *FloatPrecision* property of the *FormatLocale* object your application is using. If the *FormatSpec* object represents a double float type, this property defaults to the value of the *DoubleFloatPrecision* property of the *FormatLocale* object your application is using. See the *FormatLocale* object for more information.
Availability	Local, full, lightweight, and independent clients. Independent clients can only access *FormatSpec* objects through monitor

	points in permanent monitor sets.
Syntax	*precisionValue* = *object*.**Precision** **Element** — **Description** *precisionValue* — The precision of the configuration property value. This element has a range between 0–17. *object* — The *FormatSpec* or *ConfigProperty* object to be acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ProgramId

Summary	Contains the program ID of the *FormatSpec* object. The program ID is stored as a unique 16-digit hexadecimal number in the following format: FM:MM:MM:CC:CC:UU:TT:NN. **Note**: You may not use colons when writing the program ID. For a device with a program ID of 80:00:01:01:28:80:04:02, you can write 8000010128800402 in this property. Each *FormatSpec* object references a type. OpenLNS uses the *ProgramId* property in conjunction with the *Scope* property of the applicable *FormatSpec* object to determine which resource file contains the type's definition. For *FormatSpec* objects, the *Scope* property will only be used to identify the correct type if the *FormatType* property is set to **lcaFormatTypeNamed (0)**, which indicates that the *FormatSpec* object represents a user-defined type. In this case, the user-defined type to use within the resource file referenced by this property is identified by the *FormatName* property. If the *FormatType* property is not set to **lcaFormatTypeNamed (0)**, OpenLNS will use the Index property to determine the type associated with the *FormatSpec* object. The data stored in the *ProgramId* property of the *FormatSpec* object varies, depending on the sort of object that the *FormatSpec* applies to. • If the *FormatSpec* applies to a data point acquired through a network variable that is using a standard type, or to a monitor point that is being used to monitor a standard network variable, the *ProgramId* property will be set to 0000000000000000. • If the *FormatSpec* applies to a data point acquired through a network variable that is using a user-defined type, or to a monitor point that is being used to monitor a user-defined network variable, the *ProgramId* property

	will be set to match the program ID of the device containing the monitor point or data point. • If the *FormatSpec* object applies to a data point obtained through a ConfigProperty object, the *ProgramId* property will be set to match the program ID of the resource file that defines the configuration property type. • If the *FormatSpec* object applies to a message monitor point, the *ProgramId* property will be set to 0000000000000000.
Availability	Local, full, and lightweight clients.
Syntax	*programIdValue* = *formatSpec*.**ProgramId** **Element** — **Description** *programIdValue* — The length of network variables (in bytes) using the type referenced by this *FormatSpec* object. *formatSpec* — The *FormatSpec* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Scope

Summary	Each *FormatSpec* object references a type defined in a LonMark resource file. This property indicates the scope of the resource file that contains that type's definition. The scope of a resource file determines which devices will use the types defined in that resource file.
Availability	Local, full, lightweight, and independent clients. Independent clients can only access *FormatSpec* objects through monitor points in permanent monitor sets.
Syntax	*scopeValue* = *specObject*.**Scope** **Element** — **Description** *scopeValue* — The scope of the device resource file containing the definition of this type. The possible values of this element, which are stored in the *ConstResourceScope* constant, are as follows: **0 lcaResourceScopeStandard** This scope applies to all devices. **1 lcaResourceScopeClass** This scope applies to all devices of a specified device class from any manufacturer.

		2 lcaResourceScopeSubclass This scope applies to all devices of a specified device class and device subclass from any manufacturer. **3 lcaResourceScopeMfg** This scope applies to all devices of a specified manufacturer. **4 lcaResourceScopeMfgClass** This scope applies to all devices of a specified manufacturer and device class. **5 lcaResourceScopeMfgSubClass** This scope applies to all devices of a specified manufacturer, device class and device subclass. **6 lcaResourceScopeMfgModel** This scope applies to all devices of a specified manufacturer, device class, device subclass and model. **-1 lcaResourceScopeUnknown** The scope of the resource file is not known, or could not be found. **-2 lcaResourceScopeAutoDetermination** This value applies to the *Mode* property of *LonMarkObject* objects only. Select this value to have LNS determine the value of the *Mode* property for the *LonMarkObject* automatically. If you select this value, OpenLNS will iterate through all the available resource files from most specific to most general (highest scope to lowest scope) until it finds the functional profile template resource file containing the *LonMarkObject* object's definition. It will then assign the proper value to the *Mode* property. If OpenLNS is unable to determine the proper scope value, it will set the *Mode* property to **lcaResourceScopeUnknown (-1).**
	specObject	The *FormatSpec* object to be acted on.
Data Type	Short.	

Read/Write	Read/write.
Added to API	LNS Release 3.0.

Units

Summary	Indicates the name of the units that apply to values using this format specification.
Availability	Local, full, lightweight, and independent clients. Independent clients can only access *FormatSpec* objects through monitor points in permanent monitor sets.
Syntax	*units* = *fsObject*.**Units** **Element** — **Description** *units* — The name of the units. *fsObject* — The *FormatSpec* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

UnitsAdder

Summary	Determines how the scaled value of the data point or monitor point will be unit-converted for display as a formatted value when your application reads the *FormattedValue* propery. When your application reads the *FormattedValue* property of a data point or monitor point, the OpenLNS Object Server converts the scaled value of the data point or monitor point so that it can be display as a formatted value. To unit-convert the value, OpenLNS first multiplies it by the value in the *UnitsMultiplier* property. It then adds the value stored in the *UnitsAdder* property. The resulting value will be appropriate for the *units* that apply to this format specification. Different data formats for the same data type will be on different scales. For example, the **SNVT_temp_f** data type contains the temperature in degrees Celcius. In order to display the temperature in Fahrenheit, you would have to multiply the value by 1.8 (9/5), and then add 32.
Availability	Local, full, lightweight, and independent clients. Independent clients can only access *FormatSpec* objects through monitor points in permanent monitor sets.
Syntax	*units* = *fsObject*.**UnitsAdder** **Element** — **Description** *unitsAdder* — The value to be added. *fsObject* — The *FormatSpec* object to be acted on.

Data Type	Single.
Read/Write	Read only.
Added to API	LNS Release 3.0.

UnitsMultiplier

Summary	Determines how the scaled value of the data point or monitor point will be unit-converted for display as a formatted value when your application reads the *FormattedValue* propery. When your application reads the *FormattedValue* property of a data point or monitor point, the OpenLNS Object Server converts the scaled value of the data point or monitor point so that it can be display as a formatted value. To unit-convert the value, LNS first multiplies it by the value in the *UnitsMultiplier* property. It then adds the value stored in the *UnitsAdder* property. The resulting value will be appropriate for the *units* that apply to this format specification. Different data formats for the same data type will be on different scales. For example, the **SNVT_temp_f** data type contains the temperature in degrees Celcius. In order to display the temperature in Fahrenheit, you would have to multiply the value by 1.8 (9/5), and then add 32.
Availability	Local, full, lightweight, and independent clients. Independent clients can only access *FormatSpec* objects through monitor points in permanent monitor sets.
Syntax	*unitsMultiplier* = *fsObject*.**UnitsMultiplier** **Element** — **Description** *unitsMultiplier* — The value to be multiplied. *fsObject* — The *FormatSpec* object to be acted on.
Data Type	Single.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Interface

An *Interface* object represents an application device's external interface, which is the logical interface to a device. A device's interface specifies the number and types of functional blocks; number, types, directions, and connection attributes of network variables; and the number of message tags. The *Interface* object can be used both in the context of a particular program (part of the *DeviceTemplate* object), a particular application device (part of the *AppDevice* object), or a particular network services device (contained within an *Interfaces* collection).

The *Interface* objects contained by an application device may be a device's main interface, or they may be custom interfaces that have been added to the device. The main interface of a device can be accessed through the device's *Interface* property. Main interfaces are static

interfaces that cannot be modified. The custom interfaces that have been added to a device can be accessed through the device's *Interfaces* property. In many cases, you can modify these interfaces by adding or removing objects from their *NetworkVariables*, *LonMarkObjects*, and *MessageTags* collections.

Some of the information returned by the properties of each *Interface* object apply to the device as a whole, and not to the specific interface represented by the *Interface* object. For example, the *StaticNvCount* property indicates the number of network variables that are statically defined on the device containing an *Interface* object, and the *MaxNvInUse* property indicates the current maximum network variable index in use on the device containing an *Interface* object. In addition, the *SupportsDynamicNvsOnStaticLMOs* property indicates whether dynamic network variables can be added to the static *LonMarkObjects* on the device. Also, the *DynamicLonMarkObjectCapacity* property indicates how many dynamic *LonMarkObjects* can be added to all the interfaces on the device, not to a single interface on the device.

The *Name* property assigned to all custom interfaces on a given device must be unique. For an Interface object that represents a device's main interface, the *Name* property is initialized as an empty string.

The following table summarizes the *Interface* object.

Description	An application device's external interface.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object. *DeviceTemplate* object. *Interfaces* object.
Default Property	*Name*.
Methods	• *AddCompatibleNv* • *AddCompatibleNvEx* • *AddComplementaryNv* • *AddComplementaryNvEx* • *AddNvFromString* • *CreateTypeSpec* • *DownloadConfigProperties* • *RemoveNv* • *UploadConfigProperties*
Properties	• *ClassId* • *ConfigProperties* • *ConfigPropertiesAvailable* • *CpByHandle* • *DynamicLonMarkObjectCapacity* • *DynamicMessageTag* • *LonMarkObjects* • *MaxNvInUse* • *MaxNvSupported* • *MessageTags* • *Name* • *NetworkVariables* • *Parent* • *StaticNvCount* • *SupportsDynamicNvsOnStaticLMOs*

	• *Version*

Methods

The *Interface* object contains the following methods:

- *AddCompatibleNv*
- *AddCompatibleNvEx*
- *AddComplementaryNv*
- *AddComplementaryNvEx*
- *AddNvFromString*
- *CreateTypeSpec*
- *DownloadConfigProperties*
- *RemoveNv*
- *UploadConfigProperties*

AddCompatibleNv

Summary	Creates a *NetworkVariable* object that is compatible with a specified source network variable, and adds the new network variable to the specified *Interface* object. This method should only be called on a custom *Interface* object that is contained by a device that supports dynamic network variables.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *interfaceObject*.**AddCompatibleNv**(*nvName, sourceNvObj*) **Element** — **Description** *nvObject* — The compatible network variable to be returned. A compatible network variable is one that is of the same type and direction as the source network variable. This method is used to construct an interface object for a host application. *interfaceObject* — The *Interface* object to be acted on. *nvName* — A String containing the name of the added network variable. The name specified in this parameter can be a maximum of 16 characters, and it must not start with a number. In addition, it cannot include square brackets ([]), commas (,), or periods (.). Once the *NetworkVariable* object has been created, it may be renamed subject to the restrictions of the *Name* property. The name specified in this method may subsequently be retrieved using

		the *ProgrammaticName* property.
	sourceNvObj	The *NetworkVariable* object which the new network variable is based upon.
Added to API	Prior to LNS Release 3.0.	

AddCompatibleNvEx

Summary	Creates a *NetworkVariable* object that is compatible with a specified source network variable and adds the new network variable to the specified *Interface* object. This method differs from *AddCompatibleNv* method in that this method allows you to set options.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *interfaceObject*.**AddCompatibleNvEx**(*nvName, sourceNvObj, flags*)

Element	**Description**
nvObject	The compatible network variable to be returned. A compatible network variable is one that is of the same type and direction as the source network variable. This method is used to construct an interface object for a host application.
interfaceObject	The *Interface* object to be acted on.
nvName	A String containing the name of the added network variable. The name specified in this parameter can be a maximum of 16 characters, and it must not start with a number. In addition, it cannot include square brackets ([]), commas (,), or periods (.). Once the *NetworkVariable* object has been created, it may be renamed subject to the restrictions of the *Name* property. The name specified in this method may subsequently be retrieved using the *ProgrammaticName* property.
sourceNvObj	The *NetworkVariable* object which the new network variable is based upon.
flags	The options which will apply to the newly created network variable. The possible values for the *flags* parameter, which are contained in the

	ConstLNSInterfaceCompNvEx constant, are as follows: **0 lcaLNS_PollPassthrough** The created network variable will have the same value in *IsPolled* as the network variable it was based on. **1 lcaLNS_PollToggle** The created network variable will have the opposite value in *IsPolled* as the network variable it was based on. **2 lcaLNS_PollClear** The created network variable will have polling disabled. **3 lcaLNS_PollSet** The created network variable will have polling enable.
Added to API	Prior to LNS Release 3.0.

AddComplementaryNv

Summary	Creates a *NetworkVariable* object that is complementary to a specified target network variable, and adds the complementary network variable to the specified *Interface* object. A network variable is complementary to another network variable if it uses the same type and configuration (for example, service type, authentication setting, default value, polling attribute) but opposite direction (two network variables that may be bound together). This method is used to construct an *Interface* object for a host application. This method is typically used to create an input network variable that is subsequently bound to a target output network variable that is to be monitored This method should only be called on a custom *Interface* object that is contained by a device that supports dynamic network variables.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *interfaceObject*.**AddComplementaryNv**(*nvName, sourceNvObj*) **Element** — **Description** *nvObject* — The compatible network variable to be returned. A compatible network variable is one that is of the same type and direction

		as the source network variable. This method is used to construct an interface object for a host application.
	interfaceObject	The *Interface* object to be acted on.
	nvName	A String containing the name of the added network variable. The name specified in this parameter can be a maximum of 16 characters, and it must not start with a number. In addition, it cannot include square brackets ([]), commas (,), or periods (.). Once the *NetworkVariable* object has been created, it may be renamed subject to the restrictions of the *Name* property. The name specified in this method may subsequently be retrieved using the *ProgrammaticName* property.
	sourceNvObj	The *NetworkVariable* object for which a complementary network variable is created.
Added to API	Prior to LNS Release 3.0.	

AddComplementaryNvEx

Summary	Creates a *NetworkVariable* object that is complementary to a specified target network variable, and adds the complementary network variable to the specified *Interface* object. This method differs from *AddComplementaryNv* method in that it allows you to set options. A network variable is complementary to another network variable if it uses the same type and configuration (for example, service type, authentication setting, default value, polling attribute) but opposite direction (two network variables that may be bound together). This method is used to construct an *Interface* object for a host application. This method is typically used to create an input network variable that is subsequently bound to a target output network variable that is to be monitored This method should only be called on a custom *Interface* object that is contained by a device that supports dynamic network variables.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *interfaceObject*.**AddComplementaryNv**(*nvName, sourceNvObj*)

	Element	Description
	nvObject	The compatible network variable to be returned. A compatible network variable is one that is of the same type and direction as the source network variable. This method is used to construct an interface object for a host application.
	interfaceObject	The *Interface* object to be acted on.
	nvName	A String containing the name of the added network variable. The name specified in this parameter can be a maximum of 16 characters, and it must not start with a number. In addition, it cannot include square brackets ([]), commas (,), or periods (.). Once the *NetworkVariable* object has been created, it may be renamed subject to the restrictions of the *Name* property. The name specified in this method may subsequently be retrieved using the *ProgrammaticName* property.
	sourceNvObj	The *NetworkVariable* object for which a complementary network variable is created.
	flags	The options which will apply to the newly created network variable. The possible values for the *flags* parameter, which are contained in the *ConstLNSInterfaceCompNvEx* constant, are as follows: **0 lcaLNS_PollPassthrough** The created network variable will have the same value in *IsPolled* as the network variable it was based on. **1 lcaLNS_PollToggle** The created network variable will have the opposite value in *IsPolled* as the network variable it was based on. **2 lcaLNS_PollClear** The created network variable will have polling disabled. **3 lcaLNS_PollSet**

	The created network variable will have polling enable.
Added to API	Prior to LNS Release 3.0.

AddNvFromString

Summary	Adds a network variable to the interface using the string description created by the *ToString* method. The *NetworkVariable* object's *ToString* method exports the characteristics of a network variable to a text string. This string can be used to create a new network variable object using this method. Only unaltered strings created using the *ToString* method of the *NetworkVariable* object should be used as an argument to this method. Creating or editing these strings is not supported. Attempting to call this method on an invalid string will result in an error
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *interfaceObject*.**AddNvFromString** *nvCsvString* **Element** — **Description** *nvObject* — The compatible network variable to be returned. A compatible network variable is one that is of the same type and direction as the source network variable. This method is used to construct an interface object for a host application. *interfaceObject* — The *Interface* object to be acted on. *nvCsvString* — A String describing the network variable.
Added to API	Prior to LNS Release 3.0.

CreateTypeSpec

Summary	Creates a new **TypeSpec** object that can be modified and passed as a parameter to the following: • The **LonMarkObjects.AddFromTypeSpec** method (when creating a new dynamic *LonMarkObject* [functional block]). • The **TypeSpec** property of an existing *NetworkVariable* object to change its type (if supported). Changing the type of a *LonMarkObject* by setting its **TypeSpec** property is *not* supported.
Availability	Local, full, and lightweight clients.

Syntax	*newTypeSpec* = *interface*.**CreateTypeSpec**(*objectType*) **Element** — **Description** *newTypeSpec* — *newTypeSpec* is the newly created **TypeSpec** object. *interface* — The *Interface* object to be acted on. *objectType* — The type of the **TypeSpec** object to be created. The possible values for this element, which are specified in the *ConstTypeSpecObjectType* constant, are as follows: **0 lcaTypeSpecNv** The type applies to network variables. **2 TypeSpecLmo** This type applies to LonMark objects or functional blocks (SFPTs and UFPTs). **2 TypeSpecFb** An alias for **TypeSpecLmo**. **Note**: An appropriate exception will be thrown if the object type is not valid. The newly created **TypeSpec** object will be initialized with its **ProgramId** property set to match the **ProgramId** property of the *AppDevice* or *DeviceTemplate* object that contains the *Interface* object. All of the **TypeSpec** object's properties should be changed before calling the **Lookup** method.
Added to API	OpenLNS.

DownloadConfigProperties

Summary	Downloads the configuration property values stored in an *AppDevice* object into the physical application device associated with the *AppDevice* object. There is no functional difference between calling the *DownloadConfigProperties* method on a specific *AppDevice*, and calling the *DownloadConfigProperties* method on the *Interface* property contained by that *AppDevice*. This method requires that the device being loaded be in a configured state. Furthermore, if you are setting the configuration property values to their default values (the *downloadOptions* element sets the

	lcaConfigPropOptSetDefaults option), this method requires that the configuration property values be uploaded from the device (see the *UploadConfigProperties* method) or imported from the device's external interface file (see the *Import* method).
Availability	Local, full, and lightweight clients.
Syntax	*object*.**DownloadConfigProperties**(*downloadOptions*)

Element	Description
object	The *AppDevice* object from which configuration properties are to be downloaded into a physical device.
downloadOptions	An Integer value specifying the download options. These values can be ORed together; however, you must set the **lcaConfigPropOptLoadValues (1)** option for the values to be downloaded into the physical device. The values for this element, which are stored in the *ConstConfigPropOptions* constant, are as follows: **0** **lcaConfigPropOptLoadDefinitions** This option has no effect during a download. If this is the only option set when you call DownloadConfigProperties, OpenLNS will use the **lcaConfigPropOptLoadValues (1)** as the download option. **1** **lcaConfigPropOptLoadValues** Downloads all known values in the *AppDevice* object in the OpenLNS database to the physical device on the network. You must select this value if you want the values to be downloaded into the physical device, regardless of which other flags you set. However, this is the default option used if **lcaConfigPropOptLoadDefinitions (0)** is the only option specified in the call to *DownloadConfigProperties*, or if no flags are specified. **2**

		lcaConfigPropOptSetDefaults Sets the configuration property values in the *AppDevice* object to the default configuration property values stored in the *DeviceTemplate*, and downloads any changed values to the physical device on the network (if the **lcaConfigPropOptLoadValues** flag is set). Manufacturing-only configuration properties are not affected by this option unless it is ORed with **lcaConfigPropOptIncludeMfgOnly (8)**. This option should not be ORed with **lcaConfigPropOptLoadUnknown (4)**. Note that setting SCPTnvType configuration properties to their default values may cause the download operation to fail. To exclude those configuration properties from a download, you should OR this value with the **lcaConfigPropOptExcludeNvTypeDefaults (512)** value. **4** **lcaConfigPropOptLoadUnknown** Sets all unknown configuration property values in the *AppDevice* object to the default values stored in the *DeviceTemplate*, and then downloads all known values into the physical device (if the **lcaConfigPropOptLoadValues** flag is set). This will not effect manufacturing only configuration propertys unless ORed with **lcaConfigPropOptIncludeMfgOnly (8)**. This option should not be ORed with **lcaConfigPropOptSetDefaults (2)**, as that would override this option by setting all properties to their defaults. Note that setting SCPTnvType configuration properties to their

		default values may cause the download operation to fail. To exclude those configuration properties from a download, you should OR this value with the **lcaConfigPropOptExcludeNvTypeDefaults (512)** value. **8** **lcaConfigPropOptIncludeMfgOnly** Use this flag to in conjunction with the **lcaConfigPropOptSetDefaults (2)** and **lcaConfigPropOptLoadUnknown (4)** options if you want the download operation to include manufacturing-only configuration properties. Generally, these configuration properties should only be modified during the manufacturing process. However, OpenLNS will not enforce this requirement during a download if this flag is set, since LNS may be the tool used to set the configuration property values during the manufacturing process. **16** **lcaConfigPropOptSetUnknown** Sets all configuration properties in the *AppDevice* object in the OpenLNS database to unknown. This has no effect on the values stored in the physical device on the network. **32** **lcaConfigPropOptSetMfgOnlyUnknown** Sets all manufacturing only configuration properties to unknown. The values will not be downloaded into the physical device. **64** **lcaConfigPropOptExcludeDeviceSpecific** Downloads only those configuration properties that do not have the device-specific attribute set into the device. For example, if this option is

		ORed with **lcaConfigPropOptLoadUnknown (4)** and **lcaConfigPropOptLoadValues (1)**, LNS would set all configuration properties that are not device-specific, and whose values are unknown to their default values, and then download those values into the device. This option should not be ORed with **lcaConfigPropOptOnlyDeviceSpecific (128)**. **128 lcaConfigPropOptOnlyDeviceSpecific** Downloads only device-specific configuration properties into the device. For example, if this option is ORed with **lcaConfigPropOptLoadUnknown (4)** and **lcaConfigPropOptLoadValues (1)**, OpenLNS would set all device-specific configuration properties whose values are unknown to their defaults, and download those values into the device. This option should not be ORed with **lcaConfigPropOptExcludeDeviceSpecific (64).** **256 lcaConfigPropOptClearUpdatePending** Clears the update pending flag on the device configuration. This value may be used alone, or ORed with other values. If used alone, it will clear the update pending flag of all configuration property values in the device. It may be combined with **lcaConfigPropOptOnlyDeviceSpecific (128)** to clear only the pending flags of device specific values. This value has no effect on the database operations initiated by other flags passed to the *DownloadConfigProperties* method.

		However, updates to the device initiated by the other flags will be cancelled. For example, when combining this value with **lcaConfigPropOptSetDefaults (2)**, the operation will set the values in the database to their defaults, but the pending update flag on the device will be cleared. As a result, those values will not be loaded into the device. **512 lcaConfigPropOptExcludeNvType Default** You can OR this value with either the **lcaConfigPropOptSetDefaults (2)** or **lcaConfigPropOptLoadUnknown (4)** values to prevent setting SCPTnvType configuration property values to their defaults during a download. Setting a SCPTnvType configuration property to its default value may not be allowed due to connection constraints, and therefore would cause the download operation to fail. When ORed with **lcaConfigPropOptSetDefaults (2)**, all configuration properties other than SCPTnvType configuration properties will be set to their default values. When ORed with **lcaConfigPropOptLoadUnknown (4)**, all unknown configuration properties whose values are unknown will be set to their default values, except SCPTnvType configuration properties. This option does not affect the **lcaConfigPropOptLoadValues (1)** value. If the **lcaConfigPropOptLoadValues (1)** value is specified, SCPTnvType configuration properties with known values will be propagated to the device, even if setting their values to the default has been excluded.

Added to API	Prior to LNS Release 3.0.

RemoveNv

Summary	Removes a network variable from the *Interface* object. When this method is called, the network variable is removed from all connections containing it. If any affected connection then has only one remaining member, the connection is removed. You can only call this method on custom *Interface* objects (*Interface* objects accessed through the *Interfaces* property of an *AppDevice* or *DeviceTemplate* object)
Availability	Local, full, and lightweight clients.
Syntax	*interfaceObject*.**RemoveNv** *nameIndex* **Element** — **Description** *interfaceObject* — The *Interface* object to be acted on. *indexName* — A Long value specifying the collection index of the *NetworkVariable* object to remove, or a String value specifying the name of the *NetworkVariable* object to remove.
Added to API	Prior to LNS Release 3.0.

UploadConfigProperties

Summary	Uploads all configuration property values from a physical device on the network into the associated *AppDevice* object in the OpenLNS database. A configuration property's value is stored in a *DataPoint* object. The *ConfigProperties* collection containing all the configuration properties on a device is accessed through the *AppDevice* object's *Interface* property (*AppDevice.Interface*). The *UploadConfigProperties* method can be invoked using either the *AppDevice* object or the *Interface* object. You cannot call the *UploadConfigProperties* method on a device until you commission it with the *Commission* method.
Availability	Local, full, and lightweight clients.
Syntax	*object.UploadConfigProperties options* **Element** — **Description** *object* — The *AppDevice* object in the OpenLNS database to which configuration properties are to be uploaded. *options* — An Integer value specifying the desired upload options. The values for this element, which are stored in the *ConstConfigPropOptions* constant, are as follows:

	0 lcaConfigPropOptLoadDefinitions Reads the template file and loads the configuration property definitions for the device into the OpenLNS database if the configuration property template file on the device has not been imported or uploaded into the OpenLNS database. **1 lcaConfigPropOptLoadValues** Uploads all configuration property values from the physical device on the network to the associated *AppDevice* object in the OpenLNS database. When combined with **lcaConfigPropOptExcludeDeviceSpecific (64)**, configuration properties with the device-specific attribute set will be excluded from the upload. **2 lcaConfigPropOptSetDefaults** Sets the values stored for the device in the OpenLNS database as the default configuration property values in the *AppDevice* object's *DeviceTemplate*. This operation will change the default values that could be applied to any device using the *DeviceTemplate*. Note that this option will upload values from the physical device, regardless of whether or not it is ORed with **lcaConfigPropOptLoadValues (1)**. In either case, OpenLNS will upload all the configuration property values from the device into the OpenLNS database, and then set all the values in the database as the defaults. You can OR this option with **lcaConfigPropOptLoadUnknown (4)**. In this case, only values that are unknown in the OpenLNS database will be uploaded from the physical device. Following that, all the values stored in the OpenLNS database for the device will be set as the defaults in the *DeviceTemplate* object. This includes the values uploaded by the call to *UploadConfigProperties*, as well as all values that were known in the OpenLNS database before the operation began. **4 lcaConfigPropOptLoadUnknown** This option must be ORed with the **lcaConfigPropOptLoadValues (1)** or the **lcaConfigPropOptSetDefaults (2)** values to have any effect. You can OR this with the **lcaConfigPropOptLoadValues (1)** value to

	upload all unknown values in the OpenLNS database from the physical device on the network. Alternatively, you can OR this with the **lcaConfigPropOptSetDefaults (2)** values to upload all the unknown values into the OpenLNS database, and then set the uploaded values (as well as all values that were known in the database before the upload) as the device's defaults in the OpenLNS database. Note that all configuration properties in an *AppDevice* object start in the unknown condition. Values that have been explicitly set in the database are not affected by this option. **64** **lcaConfigPropOptExcludeDeviceSpecific** Do not upload configuration properties with the *device-specific* attribute set into the OpenLNS database. For example, if this option is ORed with **lcaConfigPropOptLoadValues (1)**, OpenLNS would upload all configuration properties that are not device-specific from the device into the OpenLNS database. This option should not be ORed with lcaConfigPropOptOnlyDeviceSpecific (128). **128** **lcaConfigPropOptOnlyDeviceSpecific** Only upload configuration properties with the *device-specific* attribute set into the OpenLNS database. For example, if this option is ORed with **lcaConfigPropOptLoadUValues (1)**, LNS would upload all device-specific configuration properties from the device into the OpenLNS database. This option should not be ORed with **lcaConfigPropOptExcludeDeviceSpecific (64)**.
Added to API	Prior to LNS Release 3.0.

Properties

The *Interface* object contains the following properties:

- *ClassId*
- *ConfigProperties*
- *ConfigPropertiesAvailable*
- *CpByHandle*
- *DynamicLonMarkObjectCapacity*
- *DynamicMessageTag*
- *LonMarkObjects*
- *MaxNvInUse*

- *MaxNvSupported*
- *MessageTags*
- *Name*
- *NetworkVariables*
- *Parent*
- *StaticNvCount*
- *SupportsDynamicNvsOnStaticLMOs*
- *Version*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Interface* object in the *ConstClassIds* constant: **19 lcaClassIdInterface** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ConfigProperties

Summary	Contains the *ConfigProperties* collection object associated with the specified *Interface* object.
Availability	Local, full, and lightweight clients.
Syntax	*configPropsCollection* = *interfaceObject*.**ConfigProperties** **Element** — **Description** *configPropsCollection* — The returned configuration properties collection. *interfaceObject* — The *Interface* object to be acted on
Data Type	*ConfigProperties* collection object.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

ConfigPropertiesAvailable

Summary	Indicates whether configuration property definitions are available for the device containing this *Interface* object.
Availability	Local, full, and lightweight clients.
Syntax	*configPropsValue* = *intObject*.**ConfigPropertiesAvailable** **Element** — **Description** *configPropsValue* — Boolean value. **TRUE**. Configuration property definitions for the device containing this interface have been either uploaded from the device, or imported from an external interface file. **FALSE**. Configuration property definitions for the device are not available. *intObject* — The *Interface* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

CpByHandle

Summary	Retrieves the *ConfigProperty* object contained in the *Interface* property of an *AppDevice* object or in the *Interface* property of a *DeviceTemplate* object selected by its *Handle* property. *Interface* objects (belonging to *AppDevice* and *DeviceTemplate* objects), *LonMarkObjects*, and *NetworkVariable* objects each have a *ConfigProperties* collection that contains all of the configuration properties that apply to the object. Typically a *ConfigProperty* object is accessed through the appropriate *ConfigProperties* collection. However, in some cases, you might need to access a *ConfigProperty* without knowing the object to which it applies. This method can be used to retrieve a *ConfigProperty* defined within an *Interface* giventhe *Handle* of a *ConfigProperty* object, even if the *ConfigProperty* applies to one or more *LonMarkObjects* or *NetworkVariable* objects. For example, this method can be used to retrieve a *ConfigProperty* when processing a *ConfigProperty* change event reported by *OnNodeIntfChangeEvent*. If the configuration property does not belong to the specified Interface, this method returns an **LCA, #6 ObjectNotFound** error. To ensure that a device-based configuration property is always found, use this method on the main interface of an *AppDevice* object from the *Interface* property, and not on a custom interface from the *Interfaces* collection. To determine all the objects to which the configuration property applies, use the *AppliesTo* property of the *ConfigProperty* object.
Availability	Local, full, and lightweight clients.
Syntax	*cpObject* = *interfaceObject*.**CpByHandle**(*cpHandle*) **Element** — **Description** *cpObject* — The returned *ConfigProperty* object. *interfaceObject* — The *Interface* object containing the configuration property. *cpHandle* — The handle of the *ConfigProperty* object to be retrieved.
Data Type	*ConfigProperty* object.
Read/Write	Read only.
Added to API	OpenLNS.

DynamicLonMarkObjectCapacity

Summary	Indicates the number of dynamic *LonMarkObject* objects (functional blocks) that can be added to the device containing this interface. A functional block represents a collection of network variables and configuration properties on a device that perform a related function. For example, a digital input device with four switches could contain one functional block for each switch. In OpenLNS, functional blocks are represented by *LonMarkObject* objects. Some device interfaces support dynamic LonMarkObjects, which means that you can add them to a device interface manually with the *Add* method. In addition, you can add network variables to a dynamic *LonMarkObject* after it has been added to a device. This property returns the number of dynamic *LonMarkObject* objects that can be added to the device containing this interface. The valid range for this property is 0–4096. You can check how many total *LonMarkObject* objects currently belong to an interface by reading the *Count* property of the *LonMarkObjects* collection contained by the interface. You can add dynamic *LonMarkObjects* to the interface using the *Add* method of the interfaces *LonMarkObjects* collection.
Availability	Local, full, and lightweight clients.
Syntax	*objectCapacity* = *interface*.*DynamicLonMarkObjectCapacity* **Element** — **Description** *objectCapacity* — The number of dynamic LonMark objects (functional blocks) that can be added to the device containing this interface. *interface* — The *Interface* object to be acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

DynamicMessageTags

Summary	Returns the collection of dynamic *MessageTag* objects contained by this interface. Some custom interfaces support dynamic message tags, meaning that you can add message tags to them with the *Add* method. You cannot add dynamic message tags to a device's main interface, and you cannot add them to some custom interfaces. The *Interface* objects contained within a device's *Interfaces* property are the device's custom interfaces, and the *Interface* object contained within the device's *Interface* property is the device's main, static interface. Typically, dynamic message tags are added to the *Network Service Device*. These message tags may be bound, like static message tags, and can then be monitored by message monitor points defined on the *Network Service Device*. The message monitor point may then be used to send messages to the device, or to devices connected to the dynamic message tag. See the *Add* method of the *MessageTags* collection for more information.
Availability	Local, full, and lightweight clients.
Syntax	*mtCollection* = *object*.**DynamicMessageTags** **Element** — **Description** *mtCollection* — The returned *MessageTags* collection. *object* — The *Interface* object to be acted on
Data Type	*MessageTags* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.20.

LonMarkObjects

Summary	Contains the *LonMarkObjects* collection object associated with the specified *Interface* object.
Availability	Local, full, and lightweight clients.
Syntax	*lmObjCollection* = *interfaceObject*.**LonMarkObjects** **Element** — **Description** *lmObjCollection* — The *LonMarkObjects* collection to be returned. *interfaceObject* — The *Interface* object to be acted on.
Data Type	*LonMarkObjects* collection object.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

MaxNvInUse

Summary	Indicates the current maximum network variable index in use on the device containing this *Interface*. This figure applies to all *Interfaces* on the device, and that this value reflects both static and dynamic network variables. The value of the *MaxNvInUse* property will always be equal to the value of the *StaticNvCount* property, minus 1, in Neuron hosted applications because they do not support dynamic network variables.
Availability	Local, full, and lightweight clients.
Syntax	*maxIndex = interfaceObject.MaxNvInUse* **Element** — **Description** *interfaceObject* — The *Interface* object to be acted on. *maxIndex* — The maximum network variable index in use. • For Neuron hosted applications, possible values are between 0 to 61. • For host-based applications, possible values are between 0 to 4095.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MaxNvSupported

Summary	Indicates the maximum possible number of network variables supported by the device containing this *Interface*. This property specifies the maximum number of network variables, whether static or dynamic, that the device containing this *Interface* object may have. The number of static network variables is fixed; therefore, this property essentially puts a limit on the number of dynamic network variables (which will be equal to *MaxNvSupported* minus *StaticNvCount*). *MaxNvSupported* will always be equal to *StaticNvCount* in Neuron Chip-hosted applications, since they do not support dynamic network variables. The possible values are 0 to 62 for Neuron Chip-hosted applications, and 0 to 4096 for host applications.
Availability	Local, full, and lightweight clients.

Syntax	*maxNvValue* = *interfaceObject*.**MaxNvSupported** **Element** **Description** *interfaceObject* The *Interface* object to be acted on. *maxNvValue* The maximum number of network variables supported.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MessageTags

Summary	Returns the collection of *MessageTags* associated with the object. This property contains the static message tags that belong to the interface. The *DynamicMessageTags* property contains the collection of dynamic message tags that have been added to the interface. See the *DynamicMessageTags* property for more information.
Availability	Local, full, and lightweight clients.
Syntax	*mtCollection* = *interfaceObject*.**MessageTags** **Element** **Description** *mtCollection* The returned *MessageTags* collection. *interfaceObject* The *Interface* object to be acted on
Data Type	*MessageTags* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** **Description** *stringValue* The name of the object. *object* The object to be acted on.

Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NetworkVariables

Summary	Returns the *NetworkVariables* collection object representing the network variables in that *Interface* object.
Availability	Local, full, and lightweight clients.
Syntax	*nvCollection* = *interfaceObject*.**NetworkVariables** **Element** — **Description** *nvCollection* — The returned *NetworkVariables* collection. *interfaceObject* — The *Interface* object to be acted on
Data Type	*NetworkVariables* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

StaticNvCount

Summary	Indicates the number of network variables that are statically defined on the device that contains this *Interface*. Statically defined network variables have indexes ranging from 0 through *n*-1, where *n* is equal to the *StaticNvCount* property. These network variables cannot be removed from the device.
Availability	Local, full, and lightweight clients.
Syntax	*staticNvCountValue = interfaceObject.StaticNvCount* **Element** — **Description** *interfaceObject* — The *Interface* object to be acted on. *staticNvCountValue* — The number of static network variables as an integer.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

SupportsDynamicNvsOnStaticLMOs

Summary	Indicates whether static *LonMarkObject* objects (functional blocks) on the device containing this interface support the addition of dynamic network variables.
Availability	Local, full, and lightweight clients.
Syntax	*dynamicNVsFlag = Interface.SupportsDynamicNvsOnStaticLMOs* **Element** — **Description** *dynamicNVsFlag* — A Boolean value. **TRUE**. Static *LonMarkObject* objects (functional blocks) on the device containing this interface support the addition of dynamic network variables. This means that you can use the *AssignNetworkVariable* method to add network variables to static *LonMarkObject* objects on this interface. You can always add network variables to dynamic *LonMarkObject* objects.

	You can determine if a *LonMarkObject* is static or dynamic by reading its *IsDynamic* property. **FALSE**. The static *LonMarkObject* objects (functional blocks) on the device containing this interface do not support the dynamic addition of network variables. **Note**: The only interface on a device that contains static LonMarkObjects is the device's main interface (the *Interface* object is accessed through the device's *Interface* property). *Interface* — The *Interface* object to be acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Version

Summary	Returns the version of the *Interface* object that is being used. The value assigned to an *Interface* object's *Version* property is incremented each time you modify the interface by changing a network variable's type, or by adding or removing a dynamic network variable from the interface. All *Interface* objects on a device share the same version number. And so when any *Interface* on a device is modified in this fashion, its version number will be incremented, as will the version number of all the other *Interface* objects on the device.
Availability	Local, full, and lightweight clients.
Syntax	*version* = *intfObject*.**Version** **Element** — **Description** *version* — The returned version number string. *intfObject* — The *Interface* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Interfaces

The *Interfaces* object is a collection of *Interface* objects. This collection contains the set of custom interface objects that have been added to a device. You can use the *Add* method to create custom interfaces on any device that supports dynamic network variables, dynamic message tags, or dynamic LonMarkObjects. The following table summarizes the *Interfaces* object.

Description	Represents a collection of *Interface* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object *NetworkServiceDevice* object
Default Property	*Item* property.
Methods	• *Add* • *Remove*
Properties	• *ClassID* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *Intefaces* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	Adds a new custom *Interface* to the device containing this collection. You can create custom interfaces on any device that supports dynamic network variables, dynamic message tags, or dynamic *LonMarkObjects*.
Availability	Local, full, and lightweight clients.
Syntax	*interfaceObject* = *interfaceColl*.**Add**(*interfaceName, sourceInterfaceObj*)

Element	Description
interfaceObject	The newly defined *Interface* object.
interfacesColl	The *Interfaces* collection object.
interfaceName	A String containing the name of the interface. The name of each custom *Interface* object on a device must be unique. If you specify a name that is already being used on the device containing this collection when you call this method, the **LCA, #3**

		lcaErrDuplicateKey exception will be thrown.
	sourceInterfaceObj	The existing *Interface* object used to create the new *Interface* object. If the *sourceInterfaceObj* element is specified, the new *Interface* object will be created with the same *NetworkVariables* collection as the *Interface* object referenced as the *sourceInterfaceObj*. If the *sourceInterfaceObj* is null, a new, empty *Interface* object will be created.
Added to API	Prior to LNS Release 3.0.	

Remove

Summary	Removes an *Interface* object from the system. Removing an *Interface* object from the *Interfaces* collection also deletes their (dynamic) *LonMarkObject* objects and their associated *NetworkVariable* and (dynamic) *MessageTag* objects.	
Availability	Local, full, and lightweight clients.	
Syntax	*interfacesColl*.**Remove** *indexName* **Element**	**Description**
	interfacesColl	The *Interfaces* collection object containing the interface to be removed.
	indexName	A Long value specifying the collection index of the *Interface* object to remove, or a String value specifying the name of the *Interface* object to remove.
Added to API	Prior to LNS Release 3.0.	

Properties

The *Intefaces* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that

	some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Intefaces* object in the *ConstClassIds* constant: **20 lcaClassIdInterfaces** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an *Interface* object from an *Interfaces* collection. You can retrieve an *Interface* object from its *Interfaces* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve an *Interface* object in *Interfaces* collections with the *Name* property by passing the *Interface* object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to

	Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *Interface* object retrieved from the collection. *collObject* — The *Interfaces* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *Interface* object to be retrieved. *stringExpression* — A string type specifying the name of the *Interface* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++,

	you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

LdrfLanguage

The *LdrfLanguage* object represents the information OpenLNS has about a supported language. All properties in this object are read only. The following table summarizes the *LdrfLanguage* object.

Description	The information OpenLNS has about a supported language.
Added to API	LNS Release 3.0.
Accessed Through	*LdrfLanguages*.
Default Property	*Name* property.
Methods	None.
Properties	• *ClassId* • *Extension* • *LanguageCode* • *LanguageId* • *LdrfId* • *Name* • *Parent*

Methods

The *LdrfLanguage* object does not contain any methods.

Properties

The *LdrfLanguage* object contains the following properties:

- *ClassId*
- *Extension*
- *LanguageCode*
- *LanguageId*
- *LdrfId*
- *Name*
- *Parent*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *LdrfLanguage* object in the *ConstClassIds* constant: **83 lcaClassIdLdrfLanguage** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Extension

Summary	Contains the three character string indicating the language that this *LdrfLanguage* object represents. This three letter sting is is identical to the string entered in the *ResourceLanguageId* property.
Availability	Local, full, lightweight, and independent clients.
Syntax	*extension* = *ldrfObject*.*Extension* **Element** — **Description** *extension* — The three character string identifying the language. Network variable types, configuration

		property types, functional profiles, and enumeration types can all reference text information used to describe their name, units, and function. This text information is contained in separate *language files*. There is one language file for every language supported by a resource file set. When a language file is translated, the references contained in the network variable types, configuration property types, and functional profiles still point to the appropriate strings. The file extension of each language file depends on the language, and is one of the following: Czech csy Danish dan Dutch (Belgian) nlb Dutch (default) nld English (UK) eng English (US) enu Finnish fin French (Belgian) frb French (Canadian) frc French (default) fra French (Swiss) frs German (Austrian) dea German (default) deu German (Swiss) des Greek ell Hungarian hun Icelandic isl Italian (default) ita Italian (Swiss) its Norwegian (Bokmal) nor Polish plk Portuguese (Brazilian) ptb Portuguese (default) ptg Russian rus Slovak sky Spanish (default) esp Spanish (Mexican) esm Swedish sve
	object	The object whose *Extensions* collection is being returned.

Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

LanguageCode

Summary	Contains the 3-letter language code for the language to which the LdrfLanguage object applies. Each LdrfLanguage object contains information about a language. For example, "ENU" is used for U.S. English.
Availability	Local, full, lightweight, and independent clients.
Syntax	*ldrfCode* = *ldrfObject*.**LanguageCode** **Element** **Description**

	ldrfCode	A 3-letter identifier for the language that the LdrfLanguage object applies to. The available language codes are as follows: Czech CSY Danish DAN Dutch (Belgian) NLB Dutch (default) NLD English (UK) ENG English (US) ENU Finnish FIN French (Belgian) FRB French (Canadian) FRC French (default) FRA French (Swiss) FRS German (Austrian) DEA German (default) DEU German (Swiss) DES Greek ELL Hungarian HUN Icelandic ISL Italian (default) ITA Italian (Swiss) ITS Norwegian (Bokmal) NOR Polish PLK Portuguese (Brazilian) PTB Portuguese (default) PTG Russian RUS Slovak SKY Spanish (default) ESP Spanish (Mexican) ESM Swedish SVE
	ldrfObject	The *LdrfLanguage* object to be acted on.
Data Type	String.	
Read/Write	Read only.	
Added to API	LNS Release 3.20	

LdrfId

Summary	Contains the 32-bit language device resource file ID for the language represented by this LdrfLanguage object.
Availability	Local, full, lightweight, and independent clients.
Syntax	*ldrfId = ldrfObject.***LdrfId** **Element** **Description**

	ldrfId	The 32-bit ID representing this language. This value is not normally used by OpenLNS clients unless they directly access the Language Device Resource File API.
	ldrfObject	The *LdrfLanguage* object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.	
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.	
Syntax	*stringValue* = *object*.**Name**	
	Element	**Description**
	stringValue	The name of the object.
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read/write.	
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.	
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.	
Syntax	*parentObject* = *object*.**Parent**	
	Element	**Description**
	parentObject	The object that is the parent of the specified *object*.

	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

LdrfLanguages

The *LdrfLanguages* object is a collection of *LdrfLanguage* objects. This collection represents all the languages known by OpenLNS and the Language Device Resource File API. This list can be used in conjunction with the *ResourceLanguageId* to select a language. The following table summarizes the *LdrfLanguages* object.

Description	Represents a collection of *LdrfLanguage* objects.
Added to API	LNS Release 3.0.
Accessed Through	*ObjectServer* object
Default Property	*Item* property.
Methods	None.
Properties	• *ClassID* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *LdrfLanguages* object does not contain any methods.

Properties

The *LdrfLanguages* object contains the following properties:

- *ClassID*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to

	Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *LdrfLanguages* object in the *ConstClassIds* constant: **84 lcaClassIdLdrfLanguages** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *LdrfLanguage* object from a *LdrfLanguages* collection. You can retrieve a *LdrfLanguage* object from its *LdrfLanguages* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *LdrfLanguage* object in *LdrfLanguages* collections with the *Name* property by passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to

	Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *LdrfLanguage* object retrieved from the collection. *collObject* — The *LdrfLanguages* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *LdrfLanguage* object to be retrieved. *stringExpression* — A string type specifying the name of the *LdrfLanguage* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++,

	you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

LonMarkAlarm

The *LonMarkAlarm* object contains the alarm status information for a *LonMarkObject* (functional block). The following table summarizes the *LonMarkAlarm* object.

Description	Contains the alarm status information for a *LonMarkObject* (functional block).
Added to API	Prior to LNS Release 3.0.
Accessed Through	*LonMarkObject* object.
Default Property	None.
Methods	None.
Properties	• *AlarmType* • *ClassId* • *IndexToSNVT* • *LimitValue* • *Location* • *ObjectId* • *Parent* • *PriorityLevel* • *TimeStamp* • *Value*

Methods

The *LonMarkAlarm* object does not contain any methods.

Properties

The *LonMarkObject* object contains the following properties:

- *AlarmType*
- *ClassId*
- *IndexToSNVT*
- *LimitValue*
- *Location*
- *ObjectId*
- *Parent*
- *PriorityLevel*
- *TimeStamp*
- *Value*

AlarmType

Summary	Identifies the alarm condition represented by the *LonMarkAlarm* object.
Availability	Local, full, and lightweight clients.
Syntax	*typeValue* = *lmAlarm*.**AlarmType** **Element** — **Description** *lmAlarm* — The *LonMarkAlarm* object to be acted on. *typeValue* — The type of alarm condition represented by the *LonMarkAlarm* object. The possible values for this element, which are contained in the *ConstAlarmTypes* constant, are as follows: **0 lcaAlarmTypeNoCondition** **1 lcaAlarmTypeAlarmCondition** **2 lcaAlarmTypeTotalServiceAlarm1** **3 lcaAlarmTypeTotalServiceAlarm2** **4 lcaAlarmTypeTotalServiceAlarm3** **5 lcaAlarmTypeLowLimitClear1** **6 lcaAlarmTypeLowLimitClear2** **7 lcaAlarmTypeHighLimitClear1** **8 lcaAlarmTypeHighLimitClear2** **9 lcaAlarmTypeLowLimitAlarm1** **10 lcaAlarmTypeLowLimitAlarm2** **11 lcaAlarmTypeHighLimitAlarm1** **12 lcaAlarmTypeHighLimitAlarm2**

	255 lcaAlarmTypeNull
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *LonMarkAlarm* object in the *ConstClassIds* constant: **45 lcaClassIdLonMarkAlarm** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

IndexToSNVT

Summary	Contains the device index number of the network variable on the *LonMarkObject* (functional block) that caused the current alarm condition. You can update the *LonMarkObject* with current information by writing the **lcaLonMarkObjectRequestUpdateAlarm (4)** value to the *Request* property of the *LonMarkObject* that contains the *LonMarkAlarm*. You can determine the device index number of a *NetworkVariable* by reading its *Index* property.
Availability	Local, full, and lightweight clients.
Syntax	*deviceIndex* = *LonMarkAlarm*.**IndexToSNVT** **Element** — **Description** *deviceIndex* — The device index number of the network variable on the

		LonMarkObject that caused the current alarm condition.
	LonMarkAlarm	The *LonMarkAlarm* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

LimitValue

Summary	Represents the data limit that the *DataValue* object was compared against in determining that an alarm condition exists. The format for this property is determined by the *AlarmFormat* property contained in the *LonMarkObject* object which contains the specified *LonMarkAlarm* object.	
Availability	Local, full, and lightweight clients.	
Syntax	*limitValue* = *lmaObject*.**LimitValue**	
	Element	**Description**
	limitValue	The limit that was exceeded
	lmaObject	The *LonMarkAlarm* object to be acted on.
Data Type	Variant.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Location

Summary	Contains the value of the specified *LonMarkAlarm* object's 6-byte location as a string.	
Availability	Local, full, and lightweight clients.	
Syntax	*locationValue* = *lmaObject*.**Location**	
	Element	**Description**
	locationValue	The location as a string.
	lmaObject	The *LonMarkAlarm* object to be acted on.
Data Type	String (6 bytes).	
Read/Write	Read only.	
Added to API	See the *LonMark Application Layer Interoperability Guidelines* for more information on the location field stored in LonMark devices.	

ObjectId

Summary	Returns the ID of the *LonMarkObject* to which this *LonMarkAlarm* applies. On static interfaces, objects are numbered from 0 to *n*-1, where *n* is the number of objects in an *AppDevice*. On dynamic interfaces, the object index may be greater than or equal to the value *n* (the number of objects on the device). See the *LonMark Application Layer Interoperability Guidelines* for more information. You can use the *LonMarkObjects* collection's *ItemByIndex* method to access the *LonMarkObject* containing the network variable. You can use the value stored in this property as the *memNumVal* element when you call the *ItemByIndex* method.
Availability	Local, full, and lightweight clients.
Syntax	*objectIdValue* = *lmaObject*.**ObjectId** **Element** — **Description** *lmaObject* — The *LonMarkAlarm* object to be acted on. *objectIdValue* — The object ID of the *LonMarkObject* object to which this *LonMarkAlarm* applies.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally,

	this property exists for an object as soon as the object is added to the API.

PriorityLevel

Summary	Returns the priority level of the Alarm.
Availability	Local, full, and lightweight clients.
Syntax	*priorityLevelValue* = *lmaObject*.**PriorityLevel** **Element** — **Description** *lmaObject* — The *LonMarkAlarm* object to be acted on. *priorityLevelValue* — The priority level of the alarm as an integer. The priority level may be returned as one of the following enumerations: **0 PR_LEVEL_0** **1 PR_LEVEL_1** **2 PR_LEVEL_2** **3 PR_LEVEL_3** **4 PR_1** **5 PR_2** **6 PR_3** **7 PR_4** **8 PR_6** **9 PR_8** **10 PR_10** **11 PR_16** **-1 PR_NUL = -1**
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TimeStamp

Summary	Returns a formatted string that represents the time that the alarm occurred.
Availability	Local, full, and lightweight clients.
Syntax	*timeStampValue* = *lmaObject*.**TimeStamp** **Element** — **Description** *lmaObject* — The *LonMarkAlarm* object to be acted on. *timeStampValue* — The time at which the alarm occured as a string. This element uses the following format: **YYYY/MM/DD HH:MM:SS:MSS**

	For example, the millisecond before the end of this millennium would be: 2999/12/31 23:59:59:999.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Value

Summary	Represents the data value that caused the alarm condition. The format of this property depends upon the *AlarmFormat* of the *LonMarkObject* object containing this *LonMarkAlarm* object. By default, it returns a string of 4 hex byte values.
Availability	Local, full, and lightweight clients.
Syntax	*alarmValue* = *lmaObject*.**Value** **Element** — **Description** *alarmValue* — The value which caused the alarm. *lmaObject* — The *LonMarkAlarm* object to be acted on.
Data Type	Variant.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LonMarkObject (Functional Block)

A functional block represents a collection of network variables and configuration properties on a device that perform a related function. For example, a digital input device with four switches could contain one function block for each switch. In OpenLNS, functional blocks are represented by *LonMarkObject* objects.

Functional profiles defining standard LonMarkObjects (SFPTs) are published by the LonMark Interoperability Association. Custom LonMarkObjects can be defined with user-defined functional profile templates (UFPTs).

The following table summarizes the *LonMarkObject* object.

Description	A set of network variables and configuration properties on a device that perform a related function.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*LonMarkObjects* collection object.
Default Property	*Name*
Methods	• *AssignNetworkVariable* • *MoveToInterface* • *UnassignNetworkVariable*
Properties	• *AlarmFormat*

	• *ClassId* • *ComponentApps* • *ConfigProperties* • *Description* • *Extensions* • *FuncProfileDescription* • *FuncProfileIsDerived* • *FuncProfileName* • *FuncProfileProgrammaticName* • *Index* • *IsDynamic* • *LonMarkAlarm* • *Mode* • *Name* • *NetworkVariables* • *Parent* • *ParentInterface* • *PrincipalNv* • *ProgrammaticName* • *ReportMask* • *Request* • *SelfTestResults* • *Status* • *TypeIndex* • *TypeSpec*

Methods

The *LonMarkObject* contains the following methods.

- *AssignNetworkVariable*
- *MoveToInterface*
- *UnassignNetworkVariable*

AssignNetworkVariable

Summary	Assigns a dynamic network variable to a *LonMarkObject* object. You can assign a dynamic *NetworkVariable* to any dynamic *LonMarkObject* on the same interface. A network variable or *LonMarkObject* is considered dynamic if it was added to the device *interface*, and is not part of the *DeviceTemplate* used by that device. You can check if a *LonMarkObject* or network variable is dynamic by reading the object's *IsDynamic* property. If a *LonMarkObject* is static, you can still assign dynamic network variables to it if the *SupportsDynamicNvsOnStaticLMOs* property of the *Interface* containing the *LonMarkObject* is set to True. If you attempt to assign a network variable to a static *LonMarkObject* that does not support dynamic network variables, the **NS, #285 lcaErrNsLmobjNotDynamic** exception will be thrown.

	The network variable and *LonMarkObject* must belong to the same device interface for this method to succeed. If they do not belong to the same interface, the **LCA, #4 lcaErrInvalidOleObject** exception will be thrown. You can move a *LonMarkObject* or network variable from one interface to another with the *MoveToInterface* method. **Notes**: If you use this method to assign a static network variable to a dynamic *LonMarkObject*, the **NS, #286 lcaErrNsLmobjNvNotDynamic** exception will be thrown. In addition, a network variable can only be assigned to one *LonMarkObject* at a time. If you attempt to assign a network variable to more than one *LonMarkObject*, the **NS, #164 lcaErrNsNvmtInuse** exception will be thrown. You can use the *UnassignNetworkVariable* method to remove a network variable from a *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*LonMarkObject.AssignNetworkVariable networkVariable, memberNumber, manufacturerAssigned* **Element** — **Description** *LonMarkObject* — The *LonMarkObject* to be acted upon. *networkVariable* — The *NetworkVariable* object being assigned to the *LonMarkObject* object. *memberNumber* — A Long value indicating the member number that will be used by the network variable. The member number must be a unique value between 1–4096. *manufacturerAssigned* — A Boolean value indicating whether the member number assigned to the network variable is a manufacturer-assigned member number (True), or a LonMark-assigned member number (False).
Added to API	LNS Release 3.20.

MoveToInterface

Summary	Moves a dynamic *LonMarkObject* from one custom interface on a device to another. Each *AppDevice* on a network includes an *Interface* property that contains the device's main interface, and an *Interfaces* collection that contains the custom interfaces that have been added to the device dynamically. The interfaces each contain

	network variables and LonMarkObjects that reflect the device's functionality on the network. You can use the *MoveToInterface* method to move a dynamic network variable or *LonMarkObject* from one custom interface on a device to another. An advantage of this is that you do not have to delete the network variable or *LonMarkObject* from the first custom interface, and then add it back to the second one. Another advantage of this method is that you can use it to remove a dynamic network variable from a device's main interface. You cannot use the *Remove* method to remove a network variable from *NetworkVariables* collection on a device's main interface, even if it is a dynamic network variable. However, you can use the *MoveToInterface* method to move a dynamic network variable from the main interface to a custom interface. Once you have done so, you can remove the network variable from the custom interface, and its removal will be propagated to the main interface. The ability to move a dynamic network variable or *LonMarkObject* from one interface to another may also be useful after you have upgraded a device's interface with the *Upgrade* method. Some static network variables and LonMarkObjects that existed on the old interface, but not the new one, will be converted to dynamic and stored in a custom *Interface* object created during the upgrade. You can use this method to move those objects back to their correct interface. For more information on this, see the online help for the *Upgrade* method. If either the main interface, or an interface from another device, is specified as the new interface for the network variable or *LonMarkObject*, then the **LCA, #4 lcaErrInvalidOleObject** exception will be thrown. If you call this method on a static *LonMarkObject* or network variable, then the **LCA, #119 lcaErrInterfaceNotModifyable** exception will be thrown. You can determine if a *NetworkVariable* or *LonMarkObject* is dynamic by reading the object's *IsDynamic* property. If you attempt to call this method on a network variable that has been previously assigned to a *LonMarkObject* with the *AssignNetworkVariable* method, then the operation will fail, and the **NS, #164 lcaErrNsNvmtInUse** exception will be thrown, unless the *LonMarkObject* object the network variable has been assigned to is a member of the target interface specified as the *newInterface* element.
Availability	Local, full, and lightweight clients.

Syntax	*object.MoveToInterface newInterface* **Element** — **Description** *object* — The *LonMarkObject* to be acted upon. *newInterface* — The *Interface* object to which the object should be moved.
Added to API	LNS Release 3.20.

UnassignNetworkVariable

Summary	Removes a dynamic network variable from a *LonMarkObject* object. The network variable to be removed must have first been added to the *LonMarkObject* with the *AssignNetworkVariable* method. If you attempt to remove a network variable that has not previously been added to the *LonMarkObject*, the **NS, #108 lcaErrNsLmobjMemNotFound** exception will be thrown. If you attempt to remove a static network variable from a *LonMarkObject*, the **NS, #286 lcaErrNsLmobjNvNotDynamic** exception will be thrown. You can check if a *LonMarkObject* or network variable is dynamic by reading the object's *IsDynamic* property.
Availability	Local, full, and lightweight clients.
Syntax	*LonMarkObject.UnassignNetworkVariable networkVariable* **Element** — **Description** *LonMarkObject* — The *LonMarkObject* to be acted upon. *networkVariable* — The *NetworkVariable* object being assigned to the *LonMarkObject* object.
Added to API	LNS Release 3.20.

Properties

The *LonMarkObject* contains the following properties:

- *AlarmFormat*
- *ClassId*
- *ComponentApps*
- *ConfigProperties*
- *Description*
- *Extensions*
- *FuncProfileDescription*
- *FuncProfileIsDerived*
- *FuncProfileName*
- *FuncProfileProgrammaticName*

- *Index*
- *IsDynamic*
- *LonMarkAlarm*
- *Mode*
- *Name*
- *NetworkVariables*
- *Parent*
- *ParentInterface*
- *PrincipalNv*
- *ProgrammaticName*
- *ReportMask*
- *Request*
- *SelfTestResults*
- *Status*
- *TypeIndex*
- *TypeSpec*

AlarmFormat

Summary	Determines the format of the *LonMarkAlarm* object contained by this *LonMarkObject*. A *LonMarkAlarm* object contains two properties that determine the reading that caused the alarm (*Value*) and the limit it exceeded (*LimitValue*). Both propeties are 4-byte fields, but there is no standard format for them. This property allows you to determine how those properties will be formatted for the *LonMarkAlarm* object associated with this *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*formatValue = lmObject.AlarmFormat* **Element** — **Description** *formatValue* — A Long value identifying the format the *LonMarkAlarm* object will have. The possible values for this element, which are stored in the *ConstAlarmFormats* constant, are as follows: **0 lcaAlarmFormatBinaryHexStr** Hexadecimal binary format. This is the default value. **1 lcaAlarmFormatShort** 1-byte integer. **2 lcaAlarmFormatShortStr** 1-byte decimal string. **3 lcaAlarmFormatUshort** 1-byte unsigned integer.

	4 lcaAlarmFormatUShortStr 1-byte unsigned decimal string. **5 lcaAlarmFormatLong** 2-byte integer. **6 lcaAlarmFormatLongStr** 2-byte decimal string. **7 lcaAlarmFormatUlong** 2-byte unsigned integer. **8 lcaAlarmFormatULongStr** 2-byte unsigned decimal string. **9 lcaAlarmFormatS32** 4-byte integer. **10 lcaAlarmFormatS32Str** 4-byte decimal string. **11 lcaAlarmFormatFloat** 4-byte real. **12 lcaAlarmFormatFloatStr** 4-byte real format string. **13 lcaAlarmFormatBinary** 4 binary bytes. *lmObject* The *LonMarkObject* to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *LonMarkObject* object in the *ConstClassIds* constant: **28 lcaClassIdLonMarkObject**

	object The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ComponentApps

Summary	Contains the *ComponentApps* collection object associated with the specified LonMarkObject. The *ComponentApps* collection is a list of LNS plug-in commands that are associated with a particular object type. Note that all *LonMarkObject* objects contain a *ComponentApps* property; however, the behavior of this property is unspecified when accessed through a *LonMarkObject* object.
Availability	Local, full, and lightweight clients.
Syntax	*appsCollection* = *object*.**ComponentApps** **Element** **Description** *appsCollection* The *ComponentApps* collection to be returned. *object* The *LonMarkObject* to be acted on.
Data Type	*ComponentApps* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConfigProperties

Summary	Contains the *ConfigProperties* collection object associated with the specified *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*configPropsCollection* = *LonMarkObject*.**ConfigProperties** **Element** **Description** *configPropsCollection* The returned *ConfigProperties* collection. *LonMarkObject* The *LonMarkObject* to be acted on
Data Type	*ConfigProperties* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *LonMarkObject* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *LonMarkObject*. *object* — The *LonMarkObject* to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *LonMarkObject*. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer.
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.*Extensions* **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FuncProfileDescription

Summary	Provides a desciptive comment of the functional profile associated with the *LonMarkObject*. This property is accessed from the functional profile template file associated with the object. This property is language dependent. Set the *System* object's *ResourceLanguageId* to control the language.
Availability	Local, full, and lightweight clients.
Syntax	*description* = *object*.**FuncProfileDescription** **Element** — **Description** *object* — The *LonMarkObject* to be acted on. *description* — The returned functional profile description string.
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

FuncProfileIsDerived

Summary	Indicates whether the user-defined functional profile template (UFPT) associated with the *LonMarkObject* inherits from a standard functional profile template (SFPT).
Availability	Local, full, and lightweight clients.
Syntax	*isFuncProfileDerivedFlag* = *object*.**FuncProfileIsDerived** **Element** — **Description** *isFuncProfileDerivedFlag* — A True of False value indicating whether the functional profile associated with the functional block is derived from an SFPT. *object* — The *LonMarkObject* to be acted on.
Data Type	Boolean.
Read/Write	Read-only.
Added to API	OpenLNS.

FuncProfileName

Summary	Returns the functional profile name associated with the *LonMarkObject*. This property is accessed from the functional profile template file associated with the object. The name returned by this property is accessed from the functional profile template file associated with this object. This property is language dependent. Set the *System* object's *ResourceLanguageId* to control the language.
Availability	Local, full, and lightweight clients.
Syntax	*typeNameValue* = *object*.**FuncProfileName** **Element** — **Description** *object* — The *LonMarkObject* to be acted on. *typeNameValue* — The functional profile name to be returned.
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

FuncProfileProgrammaticName

Summary	Returns the functional profile programmatic name associated with the *LonMarkObject*. This name is accessed from the functional profile template file associated with the object. The programmatic name is the base name stored for the object; it is not language dependent like the *FuncProfileName* property.
Availability	Local, full, and lightweight clients.
Syntax	*progNameValue* = *Object*.**FuncProfileProgrammaticName** **Element** — **Description** *progName* — The functional profile programmatic name of the object. *Object* — The *LonMarkObject* to be acted on.
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

Index

Summary	Stores the device index number of the *LonMarkObject*. This is the index number assigned to the *LonMarkObject* on the device containing the object.

Availability	Local, full, and lightweight clients.
Syntax	*indexValue* = *lmObject*.**Index** **Element** **Description** *lmObject* The *LonMarkObject* to be acted on. *indexValue* The device index number of the *LonMarkObject*.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IsDynamic

Summary	Indicates whether the *LonMarkObject* is dynamic. A LonMark object is considered to be dynamic if it was manually added to the interface it belongs to, rather than being part of the static interface defined by a device template.
Availability	Local, full, and lightweight clients.
Syntax	*isDynamicFlag* = *object*.*IsDynamic* **Element** **Description** *isDynamicFlag* A True of False value indicating whether the object is dynamic. **TRUE**. The *LonMarkObject* object is dynamic. **FALSE**. The *LonMarkObject* object is static. *object* The *MessageTag* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

LonMarkAlarm

Summary	Contains the current alarm condition for the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*lmAlarm* = *lmObject*.*LonMarkAlarm* **Element** **Description** *lmAlarm* The *LonMarkAlarm* object to be returned. *lmObject* The *LonMarkObject* to be acted upon.
Data Type	*LonMarkAlarm* object.

Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Mode

Summary	Returns the scope of the functional profile template resource file that defines the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*classMode* = *lmObject*.**Mode** **Element** — **Description** *classMode* — The scope of the functional profile template resource file that defines the *LonMarkObject*. The possible values for this property, which are stored in the *ConstResourceScope* constant (and in the *ConstConfigPropertyModes* constant for LNS versions prior to LNS Release 3.20), are as follows: **0 lcaResourceScopeStandard** Applies to all devices. **1 lcaResourceScopeClass** Applies to all devices of a specified device class from any manufacturer. **2 lcaResourceScopeSubclass** Applies to all devices of a specified device class and device subclass from any manufacturer. **3 lcaResourceScopeMfg** Applies to all devices of a specified manufacturer. **4 lcaResourceScopeMfgClass** Applies to all devices of a specified manufacturer and device class. **5 lcaResourceScopeMfgSubClass** Applies to all devices of a specified manufacturer, device class and device subclass. **6 lcaResourceScopeMfgModel** Applies to all devices of a specified manufacturer, device class, device subclass and model.

		-1 lcaResourceScopeUnknown The scope of the resource file is not known, or could not be found. **-2 lcaResourceScopeAutoDetermination** Select this value to have LNS determine the value of the *Mode* property for the *LonMarkObject* automatically. If you select this value, OpenLNS will iterate through all the available resource files from most specific to most general (i.e. highest scope to lowest scope) until it finds the functional profile template resource file containing the *LonMarkObject* object's definition. It will then assign the proper value to the *Mode* property. If OpenLNS is unable to determine the proper scope value, it will set the Mode property to **lcaResourceScopeUnknown (-1).** When importing a device interface from an external interface file, previous versions of LNS would set the *Mode* property of all *LonMarkObject* objects defined in the device interface to one of two values. • It would set the *Mode* property to 0 if the *LonMarkObject* object's *TypeIndex* property was in the range of standard Functional Profile Template (FPT) indices. • It would set the *Mode* property to 3 if the *TypeIndex* property was in the range of user-defined FPT indices. LNS Release 3.20 features automatic scope determination, which means OpenLNS will now search the set of installed and cataloged resource files to find the most device-specific match for the FPT, and set the Mode property based on this determination when the device interface is imported. If no match is found, OpenLNS will set the Mode property to **lcaResourceScopeUnknown (-1)**.
	lmObject	The *LonMarkObject* to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	

Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NetworkVariables

Summary	Returns the *NetworkVariables* collection object representing the network variables in that *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*nvCollection* = *lmObject.NetworkVariables* **Element** — **Description** *nvCollection* — The returned *NetworkVariables* collection. *lmObject* — The *LonMarkObject* object to be acted on
Data Type	*NetworkVariables* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.

Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.
Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

ParentInterface

Summary	Returns the *Interface* object to which the *LonMarkObject* belongs. Each *AppDevice* on a network contains an *Interface* property containing the device's main interface, and an *Interfaces* collection containing the custom interfaces that have been added to the device dynamically. The interfaces each contain network variables and LonMarkObjects that reflect the device's functionality on the network. The *ParentInterface* property returns the *Interface* object to which the *LonMarkObject* belongs. In the case of static LonMarkObjects and network variables, this property returns the main device interface or device template the object is associated with. In the case of a dynamic *LonMarkObject* and network variables, it returns the custom interface to which the object belongs. The *ParentInterface* property is read-only. However, you can move a dynamic network variable or *LonMarkObject* from one custom interface to another with the *MoveToInterface* method.
Availability	Local, full, and lightweight clients.
Syntax	*interface* = *object*.**ParentInterface** **Element** **Description** *interface* The *Interface* object returned by the property. *object* The *LonMarkObject* to be acted upon.
Data Type	*Interface* object.
Read/Write	Read only.

Added to API	LNS Release 3.20.

PrincipalNv

Summary	Returns the principal *NetworkVariable* object associated with this *LonMarkObject* object. This information is accessed from the functional profile template file associated with this *LonMarkObject* object.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject* = *lmObject*.**PrincipalNv** **Element** — **Description** *lmObject* — The *LonMarkObject* object to be acted on. *nvObject* — The principal network variable of the *LonMarkObject* object.
Data Type	*NetworkVariable* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ProgrammaticName

Summary	Returns the programmatic name of the *LonMarkObject*. This property contains the orignal base name of the *LonMarkObject* or network variable as "programmed" in the device containing the object. • For static *LonMarkObject* objects, LNS initially acquires the name from the functional profile template file or external interface associated with the device. • For dynamic *LonMarkObject* objects, you will specify the name to use when you create the object with the appropriate *Add* method. Initially, the *Name* property, which represents the user name of the *LonMarkObject*, will also be set to the same value as the *ProgrammaticName* property. The *Name* property can be subsequently changed for all *LonMarkObject* objects, allowing OpenLNS applications to create their own identifying names for those objects. This property can be a maximum of 16 characters long, and must conform to the character restrictions defined in version 3.3 and later of the *LonMark Application Layer Interoperability Guidelines*. You may not use the following characters in the *ProgrammaticName* property: the forward slash (/), back slash (\), period (.), and colon (:).
Availability	Local, full, and lightweight clients.
Syntax	*progName* = *Object.ProgrammaticName*

	Element	**Description**
	progName	The programmatic name of the object.
	Object	The *LonMarkObject* to be acted on.
Data Type	String.	
Read/Write	Read-only for static *LonMarkObject* objects. Read/write for dynamic *LonMarkObject* objects. You can check if a *LonMarkObject* is dynamic or static by reading the *IsDynamic* property. When writing to this property, you should note that some devices, such as the SmartServer, require that all network variables within the device have a unique programmatic name. In addition, all network variables contained within a custom *Interface* object must have unique programmatic names. If you attempt to assign a duplicate programmatic name to a network variable on such a device, the operation will fail, and the **LCA, #132 lcaErrUniqueNvNameRequired** exception will be thrown.	
Added to API	Prior to LNS Release 3.0.	

ReportMask

Summary	Returns an *ObjectStatus* object which reports which LonMark status attributes are supported by this *LonMarkObject*. Some *LonMarkObject* objects do not support all of the status attributes contained in an *ObjectStatus* object. You can use the *ReportMask* property to determine which ones are not supported. If you access an *ObjectStatus* object through the *ReportMask* property, the object represents a report mask. In this case, the values of each property of the *ObjectStatus* object reflect whether or not the *LonMarkObject* supports the related status attribute. Use the *Status* property to retrieve an *ObjectStatus* object you can use to read the actual values of the supported attributes. For example, consider the *AlarmNotifyDisabled* property of the *ObjectStatus* object. The *AlarmNotifyDisabled* property contained in an *ObjectStatus* object accessed through the *Status* property of a *LonMarkObject* indicates whether alarm notification has been disabled on the *LonMarkObject*. The *AlarmNotifyDisabled* property contained in an *ObjectStatus* object accessed through the *ReportMark* property of a *LonMarkObject* indicates whether or not you can disable alarm notification on the *LonMarkObject*. You can upate the information contained in the *ObjectStatus* report mask object by writing the value **lcaLonMarkObjectRequestReportMask (5)** to the *Request* property of the *LonMarkObject*.
Availability	Local, full, and lightweight clients.

Syntax	*statusObject* = *lmObject*.**ReportMask** **Element** **Description** *statusObject* The *ObjectStatus* object returned. *lmObject* The *LonMarkObject* object to be acted on.
Data Type	*ObjectStatus* object.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

Request

Summary	Requests a change to the functionality of a *LonMarkObject*, or requests an update of the information contained in the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*lmoObject*.**Request** *requestValue* **Element** **Description** *lmoObject* The *LonMarkObject* to be acted on. *requestValue* A Long value identifying the request being made. This element identifies the action that is to be made to the *LonMarkObject*. The values for this element, which are contained in the *ConstLonMarkObjectRequest* constant, are as follows: **0 lcaLonMarkObjectRequestNormal** Returns the *LonMarkObject* to normal status after you have disabled or overridden it by writing the **lcaLonMarkObjectRequestDisable (1)** or **lcaLonMarkObjectRequestOverride (6)** values to this property. **1 lcaLonMarkObjectRequestDisabled** Disables the *LonMarkObject*. You can return the *LonMarkObject* to normal condition by writing the **lcaLonMarkObjectRequestNormal** **2 lcaLonMarkObjectRequestUpdateStatus** Requests the status of the *LonMarkObject*. In this case, the *Status* property of the object will be updated to reflect its current state.

		3 lcaLonMarkObjectRequestSelfTest Performs a self-test on the object. The *SelfTestResults* property of the *LonMarkObject* will be updated to reflect the results of the test. **4 lcaLonMarkObjectRequestUpdateAlarm** Updates the alarm status of the *LonMarkObject*. In this case, the *LonMarkAlarm* property of the *LonMarkObject* will be updated to reflect the current alarm status of the *LonMarkObject* on the network. **5 lcaLonMarkObjectRequestReportMask** Reports the status bit mask of the *LonMarkObject*. In this case, the *ReportMask* property of the *LonMarkObject* will be updated with current information. **6 lcaLonMarkObjectRequestOverride** Overrides the *LonMarkObject*. **7 lcaLonMarkObjectRequestEnable** Enables the *LonMarkObject*. **8 lcaLonMarkObjectRequestRemoveOverride** Cancels an override of the *LonMarkObject*, without changing the enabled/disabled state of the *LonMarkObject*. You can initiate an override by writing the **lcaLonMarkObjectRequestOverride (6)** value to the Request property. You can cancel an override, and re-enable the *LonMarkObject*, by writing the **lcaLonMarkObjectRequestNormal (0)** value to this property. **9 lcaLonMarkObjectRequestClearStatus** Clears the status of the *LonMarkObject*. In this case, the information contained in the *Status* property of the *LonMarkObject* will be cleared. **10 lcaLonMarkObjectRequestClearAlarm** Clears the alarm state of the *LonMarkObject*. The information contained

	in the *LonMarkAlarm* property of the *LonMarkObject* will be cleared until the next alarm condition occurs. **11 lcaLonMarkObjectRequestAlarmNotifyEnabled** Enables alarm notification for the *LonMarkObject*. **12 lcaLonMarkObjectRequestAlarmNotifyDisabled** Disables alarm notification for the *LonMarkObject*. **13 lcaLonMarkObjectRequestManualControl** Places the *LonMarkObject* under manual control. **14 lcaLonMarkObjectRequestRemoteControl** Places the *LonMarkObject* under remote control. **15 lcaLonMarkObjectRequestProgram** Reserved. **16 lcaLonMarkObjectRequestClearReset** Cancels the reset of the *LonMarkObject*. You can initiate a reset by writing the **lcaLonMarkObjectRequestReset (17)** property to the Request property. **17 lcaLonMarkObjectRequestReset** Resets the *LonMarkObject*. You can check whether or not the reset is complete by reading the object's *ResetComplete* property.
Data Type	Long.
Read/Write	*Write*-only.
Added to API	Prior to LNS Release 3.0.

SelfTestResults

Summary	Returns an *ObjectStatus* object that reports the results of a self-test request of this object. Getting the value of this object automatically runs the object's self-test. However, if the self-test takes more than 20 seconds, an exception will be raised.

	To run a self-test on an object where the result may take more than 20 seconds to complete, follow these steps: 1. Request a self-test of the *LonMarkObject* by writing the **lcaLonMarkObjectRequestSelfTest (3)** value to the object's *Request* property. 2. After the exception is raised, poll the *SelfTestInProgress* property of the *ObjectStatus* object until it returns False. 3. Check the *FailSelfTest* property of the *ObjectStatus* object to determine why the test failed. **Note**: All devices that do not comply with version 3.0 or later of the *LonMark Application-Layer Interoperability Guidelines* do not support self-tests. Some devices that comply with version 3.0 or later of the *LonMark Application-Layer Interoperability Guidelines* may not support self-tests. You can use the *ReportMask* property to determine if a device supports self-tests. If you attempt to read this property and perform a self-test on a device that does not support self-tests, an exception will be thrown.
Availability	Local, full, and lightweight clients.
Syntax	*statusObject* = *lmoObject*.**SelfTestResults** **Element** **Description** *lmoObject* The *LonMarkObject* object to be acted on. *statusObject* The *ObjectStatus* object to be returned.
Data Type	*ObjectStatus* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Status

Summary	Stores an *ObjectStatus* object which contains information reflecting the current status of the *LonMarkObject* object. Some LonMarkObjects do not support all of the status attributes reflected in an *ObjectStatus* object, and so some of the information contained in this object may not be useful. You can use the *ReportMask* property of the *LonMarkObject* to determine which status attributes are supported. The *ReportMask* property also contains an *ObjectStatus* object. However, if you access an *ObjectStatus* object through the *ReportMask* property, the object represents a report mask. In this case, the values of most properties of the *ObjectStatus* object reflect whether or not the *LonMarkObject* supports the related status attribute. For example, consider the *ResetComplete* property of the *ObjectStatus* object. • The *ResetComplete* property contained in an *ObjectStatus* object accessed through the *Status* property of a *LonMarkObject* indicates whether the most recent reset of the *LonMarkObject* has completed. • The *ResetComplete* property contained in an *ObjectStatus* object accessed through the *ReportMask* property of a *LonMarkObject* indicates whether the *LonMarkObject* can be reset. **Note**: Some properties contain the same information, whether the *LonMarkObject* represents a report mask or a status report. These properties include the *ClassId* property, the *InvalidId* property, the *InvalidRequest* property, the *ObjectId* property, the *Parent* property and the *Summary* property. You can upate the information contained in the *ObjectStatus* object accessed through the *Status* property by writing the value **lcaLonMarkObjectRequestUpdateStatus (2)** to the *Request* property of the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*statusValue = lmObject.Status* **Element** **Description** *lmObject* The *LonMarkObject* to be acted on. *statusValue* The *ObjectStatus* object returned.
Data Type	*ObjectStatus* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TypeIndex

Summary	Returns the type index of the LonMark object. Each *LonMarkObject* object uses a type defined in the functional profile template resource files. This property returns the index assigned to that type in the functional profile template resource file containing its definition. See the *LonMark Application Layer Interoperability Guidelines* for more information on the different types of functional profile templates you can associate with a *LonMarkObject*.
Availability	Local, full, lightweight, and independent clients.
Syntax	*typeIndexValue* = *lmoObject*.**TypeIndex** **Element** — **Description** *lmoObject* — The *LonMarkObject* object to be acted on. *typeIndexValue* — The type index for the specified *ConfigProperty* object.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TypeSpec

Summary	Reurns the *TypeSpec* object associated with the *LonMarkObject*. The *TypeSpec* property of a dynamic *LonMarkObject* can only be specified when the functional block is created. The *TypeSpec* property cannot be modified once the *LonMarkObject* has been created; however, the existing *TypeSpec* object from a *LonMarkObject* can be modified and used as a template for other *NetworkVariable* objects and for creating additional dynamic *LonMarkObject* objects. The *TypeSpec* object cannot be assigned back to the *TypeSpec* property of an existing *LonMarkObject*. The *TypeSpec* property for a static *LonMarkObject* including a *LonMarkObject* belonging to a device template will be inherited from the corresponding device external interface.
Availability	Local, full, and lightweight clients.
Syntax	*typeSpecObject* = *lmoObject*.**TypeSpec** **Element** — **Description** *lmoObject* — The *LonMarkObject* to be acted on. *typeSpec* — The *TypeSpec* object associated with the *LonMarkObject*.

Data Type	TypeSpec object.
Read/Write	Read only.
Added to API	OpenLNS.

LonMarkObjects

A *LonMarkObjects* object is a collection of *LonMarkObject* objects. An instance of this collection that is accessed through an *Interface* object contains the *LonMarkObject* (functional block) objects that define that interface. You can use the *ItemByIndex* method, the *Item* property, or the *ItemByProgrammaticName* method to retrieve a *LonMarkObject* from the collection. The following table summarizes the *LonMarkObjects* collection object.

Description	A collection of *LonMarkObject* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Interface* object.
Default Property	*Item*
Methods	• *Add* • *AddEx* • *AddFromTypeSpec* • *ItemByIndex* • *ItemByProgrammaticName* • *Remove* • *RemoveByIndex*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *LonMarkObjects* object contains the following methods.

- *Add*
- *AddEx*
- *AddFromTypeSpec*
- *ItemByIndex*
- *ItemByProgrammaticName*
- *Remove*
- *RemoveByIndex*

Add

Summary	Adds a dynamic *LonMarkObject* object to the custom interface containing this collection. A function block represents a collection of network variables and configuration properties on a device that perform a related function. For example, a digital input device with four switches could contain one function block for each

<table>
<tr><td></td><td>switch. In OpenLNS, function blocks are represented by LonMarkObject objects.

Some custom device interfaces support dynamic LonMarkObject objects, which means that you can add them to the interface manually. You can determine if an interface supports dynamic LonMarkObject objects by reading the interface's DynamicLonMarkObjectCapacity property. If the device interface does not support dynamic function blocks, this property will be set to 0, and use of the Add method will cause the LCA, #119 lcaErrInterfaceNotModifyable exception to be thrown.

If the device interface supports dynamic function blocks, the DynamicLonMarkObjectCapacity property will be set to a value greater than 0, and you can add function blocks to the interface with this method.

Note that you cannot add LonMarkObjects to a device's main interface, although the DynamicLonMarkObjectCapacity property of that Interface may be set to a non-zero value. The main interface of a device is stored in its Interface property. The custom interfaces of a devices are stored in its Interfaces property.

The Name and ProgrammaticName properties of the new LonMarkObject object will be set to match the name specified as the fbName element.

The name assigned to the LonMarkObject objects on each device must be unique. If you attempt to use a name that is already used on a device when adding a new LonMarkObject object, the operation will fail, and the LCA, #151 lcaErrUniqueNameRequired exception will be thrown. Echelon recommends that you make sure all LonMarkObject objects in the OpenLNS database have unique names.</td></tr>
<tr><td>Availability</td><td>newLonMarkObject = LonMarkObjects.Add fbName, fbType</td></tr>
<tr><td>Syntax</td><td>objectColl.Add name

Element — Description

newLonMarkObject — The LonMarkObject returned by the function. This is the newly defined LonMarkObject.

LonMarkObjects — The LonMarkObject collection to be acted upon.

fbName — The name of the function block to be created. This can be a maximum of 16 characters long, and must conform to the character restrictions defined in version 3.3 the LonMark Application Layer Interoperability Guidelines. These restrictions are that the name must not begin with a number, and it cannot include square</td></tr>
</table>

		brackets ([ ]), periods (.) or commas (,).
	fbType	Enter a value between 0-25,000. You can use this value to identify the functional profile template associated with the new LonMarkObject. See the *LonMark Application Layer Interoperability Guidelines* for more information on the different types of functional profile templates you can associate with a LonMarkObject.
Added to API	Prior to LNS Release 3.0.	

AddEx

Summary	Adds a dynamic *LonMarkObject* object to a custom interface containing this collection.	
Availability	Local, full, and lightweight clients.	
Syntax	*newLonmarkObject* = *lonMarkObjects*.**AddEx**(*lmoName*, *lmoType*, *programId*, *scope)*	
	Element	**Description**
	newLonMarkObject	The new *LonMarkObject* object to be created.
	lonMarkObjects	The collection of *LonMarkObjects* objects to be added to the custom interface.
	lmoName	A String containing the name of the *LonMarkObject*.
	lmoType	The FPT key of the *LonMarkObject* to be created.
	programId	The program ID of the resource file set in which the FPT is defined.
	scope	The scope of the resource file set containing the FPT definition. An appropriate exception will be thrown if the type is not found in the resource file set specified
Added to API	OpenLNS.	

AddFromTypeSpec

Summary	Creates a new *LonMarkObject* using an existing *TypeSpec* object. Validation is performed on the *TypeSpec* object through a call to the *Lookup* method of the *TypeSpec* object. An appropriate exception will be thrown if the FPT is not found, if the type of the *TypeSpec* object object is not set to

	lcaTypeSpecLmo, or it is invalid in any other way.
Availability	Local, full, and lightweight clients.
Syntax	*newLonmarkObject* = *lonMarkObjects*.**AddFromTypeSpec**(*fbName*, *lmoTypeSpec)* **Element** — **Description** *newLonMarkObject* — The new *LonMarkObject* object to be created. *lonMarkObjects* — The collection of *LonMarkObjects* objects to be added. *lmoName* — A String containing the name of the *LonMarkObject*. *lmoTypeSpec* — The **TypeSpec** object to be used as the source for the new *LonMarkObject*.
Added to API	OpenLNS.

ItemByIndex

Summary	Retrieves a *LonMarkObject* object from a *LonMarkObjects* collection. The *LonMarkObject* object to be retrieved must be specified by its index value.
Availability	Local, full, and lightweight clients.
Syntax	*itemObject* = *itemsColl.ItemByIndex index* **Element** — **Description** *itemObject* — The *LonMarkObject* retrieved from the collection. *itemsColl* — The *LonMarkObjects* collection to be acted on. *index* — An Integer value specifying the *Index* property of the *LonMarkObject* to be retrieved.
Added to API	LNS Release 3.0.

ItemByProgrammaticName

Summary	Retrieves a *LonMarkObject* object from a *LonMarkObjects* collection by its programmatic name. You can determine the programmatic name of a *LonMarkObject* by reading its *ProgrammaticName* property. Alternatively, you can retrieve a *LonMarkObject* from these collections by their user names using the *Item* property.
Availability	Local, full, and lightweight clients.

Syntax	*retrievedObject = collection.ItemByProgrammaticName progName* **Element** — **Description** *retrievedObject* — The *LonMarkObject* to be retrieved from the collection. *collection* — The *LonMarkObjects* collection being acted upon. *progName* — The programmatic name of the *LonMarkObject* to be retrieved.
Data Type	*LonMarkObject* object.
Added to API	LNS Release 3.20.

Remove

Summary	Removes a *LonMarkObject* object from the specified *LonMarkObjects* collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The *LonMarkObjects* collection containing the object to be removed. *name* — A Long value specifying the collection index of the *LonMarkObject* object to be removed, or a String value specifying the name of the *LonMarkObject* object to be removed.
Added to API	Prior to LNS Release 3.0.

RemoveByIndex

Summary	Removes a dynamic *LonMarkObject* (functional block) from the custom interface containing this collection. You can specify the *LonMarkObject* to be deleted by its device index number. You can determine the device index of a *LonMarkObject* by reading its *Index* property. **Note**: You cannot remove *LonMarkObjects* from a device's main interface. The main interface of a device is stored in its *Interface* property.
Availability	Local, full, and lightweight clients.
Syntax	*LonMarkObjects*.**RemoveByIndex** *deviceIndex, removalFlags* **Element** — **Description** *LonMarkObjects* — The *LonMarkObjects* collection to be

		acted upon.
	deviceIndex	A Long value indicating the device index number of the *LonMarkObject* to be deleted.
	removalFlags	The removal options to be used when deleting the *LonMarkObject*. These options determine whether or not network variables assigned to the *LonMarkObject* will be deleted. The valid options for the removalFlags element, which are contained in the *ConstLonMarkObjectRemoveFlags* constant, are as follows: **0 lcaLonMarkObjectRemoveObjectOnly** Leaves all member network variables assigned to the *LonMarkObject* on the interface containing the collection. OpenLNS will mark these network variables as not being assigned to a *LonMarkObject*. **1 lcaLonMarkObjectRemoveNVs** Deletes all member network variables assigned to the *LonMarkObject*. However, if you select this option and any of the network variables assigned to the LonMark object are bound, the operation will fail, and the **NS, #287 lcaErrNsLmobjInUse** exception is thrown. **3 lcaLonMarkObjectRemoveAnd DisconnectNVs** Deletes all member network variables assigned to the *LonMarkObject*. If any of the network variables are bound, they will be disconnected before they are deleted.
Added to API	LNS Release 3.20.	

Properties

The *LonMarkObjects* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *LonMarkObjects* object in the *ConstClassIds* constant: **29 lcaClassIdLonMarkObjects** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *LonMarkObject* object from a *LonMarkObjects* collection. You can retrieve a *LonMarkObject* object from its

	LonMarkObjects collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *LonMarkObject* object in *LonMarkObjects* collections that contain objects with the *Name* property by passing the object's name as a string expression
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *LonMarkObject* object retrieved from the collection. *collObject* — The *LonMarkObjects* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *LonMarkObject* object to be retrieved. *stringExpression* — A string type specifying the name of the *LonMarkObject* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Comments	The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

MessageTag

A *MessageTag* object represents a single message tag. Message tags are used for defining application message connections. The following table summarizes the *MessageTag* object.

Description	A single message tag.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*MessageTags* collection object.
Default Property	*Name*
Methods	• *AddTarget* • *Connect* • *Disconnect*
Properties	• *AddressTableIndex* • *AppDevice*

	• *AppDeviceName* • *ClassId* • *ConnectDescTemplate* • *Direction* • *Index* • *IsDynamic* • *MtHubs* • *MtTargets* • *Name* • *Parent*

Methods

The *MessageTag* object contains the following methods.

- *AddTarget*
- *Connect*
- *Disconnect*

AddTarget

Summary	Adds a single *MessageTag* to a hub's pending target list. This method defines a hub message tag's pending target list. This list is used when the *Connect* or *Disconnect* method is invoked to create or remove a message tag connection. *Connections*, as defined within OpenLNS, always consist of a single hub and one or more complementary targets. To create a message tag connection, follow these steps: 1. Select a single hub object and a set of one or more target objects to connect to the hub. 2. Add the targets to the pending target list by invoking the hub's *AddTarget* method for each target object (up to a maximum of 25, see below). 3. When the list is complete, invoke the hub's *Connect* method. To remove a connection, invoke the *Disconnect* method, instead. The pending target list will only hold 25 targets at a time, but it is cleared upon completion of the *Connect* or *Disconnect* method. You can therefore create larger connections by iterating through the process outlined above. For example, upon completion of the *Connect* method, you can add additional targets by invoking the *AddTarget* method on the original hub object. You can then invoke the *Connect* method to append the new targets to the previously defined connection. For message tags, different restrictions on multiple connections apply depending on whether a message tag is a static tag, dynamic tag, or the predefined msg_in tag.

	• A static or dynamic message tag may only appear in one connection (it may be a hub or target, but not both). A msg_in tag may appear in multiple connections. • If a static or dynamic tag is a hub, its targets may be a mix of declared tags, dynamic tags and msg_in tags. • When a msg_in tag is a hub, its targets must all be static tags or dynamic tags because the msg_in tag may only be used to receive messages. **Note**: These requirements prevent the creation of mirrored connections. In addition, neither type of message tag may be used for turnaround connections, connections where the hub and target both lie on the same application device.
Availability	Local, full, and lightweight clients.
Syntax	*nvMtObject*.**AddTarget** *targetObject* **Element** — **Description** *nvMtObject* — The hub *MessageTag* object. *targetObject* — The *MessageTag* object to be added to the target list.
Added to API	Prior to LNS Release 3.0.

Connect

Summary	Connects a hub message tag to the message tags contained in the hub's pending target list. This method creates a new connection or adds to an existing one. The connection consists of the hub and its targets. The hub is the message tag object upon which the method is invoked. Before invoking the *Connect* method, one or more targets must be added to the hub's pending target list using the *AddTarget* method. When the method is invoked, the OpenLNS Server defines the connection (using the parameters specified in the hub's *ConnectDescTemplate* object) and, if the *MgmtMode* property is set to **lcaMgmtModePropagateConfigUpdates (0)**, it connects the objects on the network, then clears the hub's pending target list. As part of the connection process, the OpenLNS Server updates the hub object's *MtTargets* property, as appropriate. If a new connection is created, the hub object is also added to the appropriate *MessageTags* property of the system's *Connections* object. When you create large or complex connections that require calling the *Connect* method more than once, you should use the *StartTransaction* and *CommitTransaction* methods to group the calls into a single transaction.

	You can use the *OnNodeConnChangeEvent* to track when connections are created or modified with this method.
Availability	Local, full, and lightweight clients.
Syntax	*MtObject*.*Connect* **Element** **Description** *MtObject* The hub *MessageTag* object to be connected.
Added to API	Prior to LNS Release 3.0.

Disconnect

Summary	Removes a hub or message tag's entire connection or disconnects the message tags contained in the hub's pending target list. The behavior of this method is dependent upon the state of the hub's pending target list, which was created by the *AddTarget* method. • If the pending target list is empty, the method disconnects all members of the connection (all message tags contained within the hub's *MtTargets* property). • If elements have been added to the pending target list, the method disconnects those particular elements and clears the pending target list. When all targets have been disconnected from a hub message tag, the connection ceases to exist. The hub is subsequently deleted from the appropriate *MessageTags* property of the system's *Connections* object. You can use the *OnNodeConnChangeEvent* to track when connections are removed with this method.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Disconnect** **Element** **Description** *object* The *MessageTag* object to be acted on.
Added to API	Prior to LNS Release 3.0.

Properties

The *MessageTag* object contains the following properties:

- *AddressTableIndex*
- *AppDevice*
- *AppDeviceName*
- *ClassId*
- *ConnectDescTemplate*
- *Direction*

- *Index*
- *IsDynamic*
- *MtHubs*
- *MtTargets*
- *Name*
- *Parent*

AddressTableIndex

Summary	Contains the address table index value assigned to the message tag. Some devices communicate with one another by sending explicit messages on a message tag. When an OpenLNS application requests that a device should share information with another device via a dynamic message tag, an address table entry is created on the device sending the information. Note that address table entries for static message tags is pre-defined. The address table entry associates the message tag with the domain/subnet/node address of the device that is to receive the information, or with a group address that identifies the set of devices to receive the information. This property contains the index value of the address table entry created for the message tag. • For static message tags, the address table index is always the same as the message tag's index value, which is stored in the message tag's *Index* property and is used to identify the message tag. • For dynamic message tags, the address table index may be different than the message tag's index value. For unbound, dynamic message tags, this property will return the value 65,535.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*index* = *messageTag*.**AddressTableIndex** **Element** — **Description** *index* — The address table index associated with the message tag. *messageTag* — The MessageTag object being acted upon.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

AppDevice

Summary	Identifies the *AppDevice* object containing this message tag.
Availability	Local, full, and lightweight clients.
Syntax	*appDevObject* = *object*.*AppDevice* **Element** **Description** *appDevObject* The *AppDevice* object to be returned. *object* The *MessageTag* object to be acted on.
Data Type	*AppDevice* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

AppDeviceName

Summary	Returns the name of an application device that contains the message tag.
Availability	Local, full, and lightweight clients.
Syntax	*nvMtName* = *nvMtObject*.**AppDeviceName** **Element** **Description** *nvMtObject* The *MessageTag* object to be acted upon. *nvMtName* The name of the message tag.
Data Type	String
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *MessageTag* object in the *ConstClassIds* constant: **22 lcaClassIdMessageTag** *object* The object to be acted on.
Data Type	Integer.

Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ConnectDescTemplate

Summary	Contains the *ConnectDescTemplate* object associated with the specified *MessageTag* object. This template contains parameters used when creating connections. Setting the *ConnectDescTemplate* property with a modified *ConnectDescTemplate* object will cause the attributes of a connection to be updated, if the message tag the property belongs to is the hub for that connection. If the message tag is not yet added to a connection, it is assumed that you will call the *Connect* method to create the connection immediately after setting this property. If you do not, reading the *ConnectDescTemplate* property will return the old (default) *ConnectDescTemplate* object. To modify the attributes of an existing connection description, modify its *ConnectDescTemplate* properties as if it were a new object. Setting the *ConnectDescTemplate* property with a modified *ConnectDescTemplate* object will cause the attributes of a connection to be updated, if the message tag the *ConnectDescTemplate* object belongs to is the hub for that connection.
Availability	Local, full, and lightweight clients.
Syntax	*cdTemplateObject* = *object*.*ConnectDescTemplate* **Element** — **Description** *cdTemplateObject* — The *ConnectDescTemplate* object. *object* — The *MessageTag* object.
Data Type	*ConfigProperties* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Direction

Summary	Specifies whether the message tag is an input or output message tag.
Availability	Local, full, and lightweight clients.
Syntax	*directionValue* = *object*.**Direction** **Element** — **Description** *directionValue* — The direction of the message tag. The vallid values for this element, which are contained in the *ConstMtDirections*

		constant, are as follows: **0 lcaMtDirectionInput** The message tag is an input message tag. **2 lcaMtDirectionOutput** The message tag is an output message tag.
	object	The *MessageTag* object.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Index

Summary	Returns the index within an application device of the message tag.	
Availability	Local, full, and lightweight clients.	
Syntax	*index* = *object*.**Index** **Element** **Description** *object* The object to be acted on. *index* Index of the message tag. The valid values for this element are 0–14. The special message tag, **msg_in**, is represented by MSG_IN_TAG (-2).	
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

IsDynamic

Summary	Indicates whether the message tag is dynamic. A message tag is considered to be dynamic if it was manually added to the interface it belongs to, rather than being part of the static interface defined by a device template.	
Availability	Local, full, and lightweight clients.	
Syntax	*isDynamicFlag = object.IsDynamic* **Element** **Description** *isDynamicFlag* A True of False value indicating whether the object is dynamic. **TRUE**. The message tag is dynamic. **FALSE**. The message tag is static.	

	object The *MessageTag* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

MtHubs

Summary	Returns a collection containing all message tag hubs for which the message tag is a target, including the message tag if it is the hub for a connection. Along with the *MtTargets* property, this property enables you to manage complex network connections involving multiple hubs and sets of targets. When a new hub is added, it will not neccessarily be added to the end of the list of hubs; therefore, you should update the cached copy of the complete hub list when you add or delete a hub.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*mtCollection* = *mtObject*.**MTHubs** **Element** **Description** *mtCollection* The returned *MessageTags* collection. *mtObject* The specified *MessageTag* object.
Data Type	*MessageTags* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

MtTargets

Summary	Returns the *MessageTags* collection object containing the message tag targets for the specified hub *MessageTag* object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*mtCollection* = *mtObject*.*MtTargets* **Element** **Description** *mtCollection* The returned *MessageTags* collection. *mtObject* The specified *MessageTag* object.
Data Type	*MessageTags* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

MessageTags

A *MessageTags* object is a collection of *MessageTag* objects. Note that the *Interface* object contains two *MessageTags* collections:

1. The *MessageTags* collections accessed through the *MessageTags* property, which contains the interface's static message tags.
2. The *MessageTags* collections accessed through the *DynamicMessageTags* property, which contains the interface's dynamic message tags.

The following table summarizes the *MessageTags* object.

Description	A collection of *MessageTag* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Connections* object. *Interface* object.
Default Property	*Item*
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *MessageTags* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	This method allows you to add message tags to any device that supports monitor sets, and use those message tags to send explicit messages from that device to a group of devices, as with static message tags. For example, consider the case of a *NetworkServiceDevice*. Network Service Devices do not contain static message tags. However, you can use this method to add dynamic message tags to the *AppDevice* object that represents a *NetworkServiceDevice*. Once you have done so, you could connect the message tag to the devices you want to send messages to. Following that, you could create a *permanent message monitor point* on the *NetworkServiceDevice* that specifies the new dynamic message tag as the monitor target. You could then open the monitor set, and use the message monitor point to send explicit messages from the *NetworkServiceDevice* to any number of devices on your network. The procedure to follow when connecting message tags and

	devices is described in the OpenLNS Programmer's Guide. The name of the new message tag (as specified with the name element) must be unique on the *Interface* object containing the *MessageTags* collection. If you attempt to add a message tag to a message tag collection on a static interface or a custom interface that does not support dynamic message tags, the **LCA, #119 lcaErrInterfaceNotModifyable** exception will be thrown. The *Interface* objects contained within a device's Interfaces property are the device's custom interfaces, and the *Interface* object contained within the device's *Interface* property is the device's main, static interface. The newly created message tags will be stored in the *MessageTags* collection accessed through the *DynamicMessageTags* property of the *Interface* object involved. The static message tags contained by the *Interface* are stored in its *MessageTags* property. Note that you can only call this method on the collection accessed through the *DynamicMessageTags* property. In addition, each device supports a total of 65,533 dynamic message tags.
Availability	Local, full, and lightweight clients.
Syntax	*tag* = *messageTags*.**Add** *name* **Element** — **Description** *messageTags* — The *MessageTags* collection object being acted upon. *name* — A String representing the name of the new message tag. The name can be a maximum of 85 characters long, and it cannot include the forward slash (/), back slash (\), period (.), and colon (:) characters. *tag* — The *MessageTag* object returned by the method. This is the newly defined message tag.
Added to API	LNS Release 3.20.

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed.

	name	A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.	

Properties

The *MessageTags* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *MessageTags* object in the *ConstClassIds* constant: **25 lcaClassIdMessageTags** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count**

	Element	**Description**
	returnValue	The number of objects in the collection as a long integer.
	object	The collection object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Item

Summary	A *MessageTags* object is a collection of *MessageTag* objects. You can retrieve a *MessageTag* object from its *MessageTags* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *MessageTag* object in *MessageTags* collections with the *Name* property by passing the *MessageTag* object's name as a string expression	
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.	
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*)	
	Element	**Description**
	retrievedObject	The *MessageTag* object retrieved from the collection.
	collObject	The *MessageTags* collection object to be acted on.
	index	A Long type specifying the ordinal index of the *MessageTag* object to be retrieved.
	stringExpression	A string type specifying the name of the *MessageTag* object to be retrieved.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.

Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

MonitorSet

A *MonitorSet* object represents a set of network variable and message tag monitor points. You can use this object to maintain and monitor all of the monitor points it contains as a group (points can also be controlled individually). This object also contains the default settings that are applied to all monitor points as they are added to the set.

There are two separate types of *MonitorSet* objects: permanent *MonitorSet* objects, which can be used in multiple client sessions, and temporary *MonitorSet* objects, which can only be used in a single client session. The rest of this section describes how you can access each type of MonitorSet object, and when you will want to use each type.

Each *Network* object contains a *MyVNI* property, which returns an *AppDevice* object representing the network service device (NSD) of a client computer on the network. You can use this *AppDevice* object to access all the *MonitorSet* objects that are stored in the OpenLNS database for your client computer's NSD. Echelon recommends that you only use the *MyVNI* property to access *MonitorSet* objects when you need to create *MonitorSet* objects, or when you need to modify the configuration of those *MonitorSet* objects. For actual monitor and control operations, you should use the *CurrentMonitorSets* property of the *Network* object.

The *CurrentMonitorSets* property returns a collection of all the *MonitorSet* objects on a network that are currently stored in your client's NSD. This may be useful if you have created monitor sets while the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**. Although those monitor sets exist in the OpenLNS database and can be accessed through the *MyVNI* property mentioned in the previous paragraph, they will not be commissioned into the NSD, and cannot be enabled or used for monitor and control operations, until the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)** and the NSD is updated. The collection accessed through the *CurrentMonitorSets* property allows access to all the monitor sets you can currently use on a network (the collection accessed through the *MyVNI* property allows access to these monitor sets, as well as those that have not yet been commissioned into your client computer's NSD). You can use all the monitor sets obtained through the *CurrentMonitorSets* property as runtime monitor sets, meaning that you can enable them and use them for monitoring operations. However, changes to their configuration are not allowed when accessed through this collection. As noted previously, you should use the collection obtained through the *MyVNI* property when you need to write to the configuration of your client's local MonitorSet objects.

You should use the permanent *MonitorSet* objects accessed through the *MyVNI* and *CurrentMonitorSets* properties when you need to create monitor points that will be used in multiple client sessions.

If you need monitor points that will only be used in a single client session, you should use temporary MonitorSet objects. You can create a temporary monitor sets with the *CreateTemporaryMonitorSet* method. Temporary monitor sets are opened automatically by OpenLNS as they are created, and they can only be accessed from the client that created them. They cannot be accessed from the permanent *MonitorSets* collections described above.

When a client releases a temporary monitor set, or when the client session in which a temporary monitor set was created ends, the temporary monitor set and all the monitor points it contains are deleted. If you need to create a group of monitor points that you can

use in multiple client sessions or that you intend to use multiple times, you should use the permanent *MonitorSet* objects described earlier in this section. However, if you do not need to re-use a monitor set, you should use temporary monitor sets, as it takes less time and network resources to create them.

The properties and methods that can be used on a temporary *MonitorSet* object and its monitor points it contains are generally the same as those that can be used on permanent *MonitorSet* object and its monitor points. However, if you have been using *MonitorSet* objects with OpenLNS versions prior to LNS Release 3.20, you should note the following exceptions to this rule:

1. Temporary *MonitorSet* objects cannot be created or used while in *independent mode*. And the *Open* and *Close* methods have no effect on temporary MonitorSet objects, because temporary MonitorSet objects are opened as soon as they are created, and closed as soon as the client session in which they were created ends. You should also note that temporary monitor sets are not enabled as they are opened. You must explicitly enable temporary monitor sets and temporary monitor points with your application using the applicable *Enable* method.
2. In addition, the *DefaultOptions* properties stored in *MsgMonitorPoint* and *NvMonitorPoint* objects in temporary monitor sets are not accessible. The values applied to these properties are taken from the temporary monitor set's *MsgOptions* or *NvOptions* properties. For more information, see the *DefaultOptions* property.
3. Monitor points in temporary monitor sets do not support the use of connection description templates to set certain monitoring options, as monitor points in permanent monitor sets do. As a result, you must set the *connDesc* element to NULL when you add a monitor point to a temporary monitor set. See the online help for the Add method for the *MsgMonitorPoints* and *NvMonitorPoints* collections for more information on this.
4. Network variable monitor points in temporary monitor sets cannot be automatically bound to the monitoring node. This means that the *UseBoundUpdates* property of all temporary monitor sets and monitor points should be set to False. For more information on this, see the online help for the *UseBoundUpdates* property.

The following table summarizes the *MonitorSet* object.

Description	A set of network variable and message tag monitor points.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*MonitorSets* object.
Default Property	*Name*.
Methods	• *Close* • *Disable* • *Enable* • *Open*
Properties	• *ClassId* • *IsEnabled* • *IsOpen* • *IsPollingEnabled* • *MsgMonitorPoints* • *MsgOptions* • *Name* • *NvMonitorPoints* • *NvOptions*

	• *Parent* • *Tag*

Methods

The object contains the following methods.

- *Close*
- *Disable*
- *Enable*
- *Open*

Close

Summary	Closes a *MonitorSet* object. For permanent *MonitorSet* objects, this method causes the monitor set to be closed, and all the monitor points (*MsgMonitorPoint* and *NvMonitorPoint* objects) in the set to be disabled. You can re-open and re-enable the monitor set later with the *Open* method. Note that it is more efficient to disable and re-enable a monitor set than to close and re-open a monitor set later in a given client session. When this method is invoked on a monitor set, the client application will be notified of the disabling of each monitor point in the set via the Object Server's *OnNvMonitorPointEvent* and *OnMsgMonitorPointEvent* events. This method does not apply to *MonitorSet* objects that were created as temporary monitor sets because all temporary *MonitorSet* objects are deleted permanently as soon as the client session in which they were created ends. You can create temporary *MonitorSet* objects with the *CreateTemporaryMonitorSet* method.
Availability	Local, full, lightweight, and independent clients. Note that temporary monitor sets are not available on Independent clients.
Syntax	*monSetObject.Close* **Element** **Description** *monSetObject* The *MonitorSet* object to be closed.
Added to API	LNS Release 3.0.

Disable

Summary	Disables monitoring of a monitor set. If monitoring is disabled for an entire monitor set, then all the monitor points in the set will be disabled. You will not be able to re-enable those monitor points until the *Enable* method is called on the monitor set. You can also disable an individual *message monitor point* or

	network variable monitor point. If you explicitly disable monitoring of a single monitor point, you can only re-enable monitoring of that monitor point by calling the *Enable* method on the monitor point.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*object*.**Disable** **Element** **Description** *object* The *MonitorSet* object to be disabled.
Added to API	LNS Release 3. 0.

Enable

Summary	Enables monitoring of a monitor set. You can use this method to enable monitoring of all network variable and message monitor points in the monitor set. If you set the *doPoll* parameter to True, all network variable monitor points in the set will be polled, unless they had polling explicitly disabled with the *Disable* method. If a *message monitor point* or *network variable monitor point* has been explicitly disabled with the *Disable* method, it will not be re-enabled when the *Enable* method is called on the monitor set. You can only re-enable that monitor point by calling the *Enable* method on it. For example, if you call the *Disable* method on a monitor point named Point A, and then call the *Enable* method on the monitor set containing Point A, Point A would not be enabled. However, all other monitor points in the monitor set would be enabled. You would need to call the *Enable* method on Point A to re-enable it. If a monitor point is disabled, and its value changes before it is re-enabled, the point will not be updated with the new value until it has been re-enabled and a monitor point update is received. You can, however, configure the network variable monitor points in your network to fetch the values of the network variables they are monitoring as soon as they are enabled. • To do this for all network variable monitor points in a monitor set, set the *GenerateInitialFetch* property of the monitor set's *NvMonitorOptions* object to True. • To do this for a single network variable monitor point, set the *GenerateInitialFetch* property of the network variable monitor point's *CurrentOptions* to True.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points

	are not available on Independent clients.
Syntax	*object*.**Enable** *doPoll* **Element** **Description** *object* The *MonitorSet* object to be enabled. *doPoll* A Boolean value. **TRUE**. Turn on polled monitoring for the network variable monitor points in the monitor set. **FALSE**. Leave polled monitoring of this monitor set turned off.
Added to API	LNS Release 3. 0.

Open

Summary	Opens a *MonitorSet* object, which causes all the monitor points in the monitor set to be instantiated. You should not open a monitor set in the same explicit transaction in which it was created, or in which monitor points were added to it. For more information on using transactions with LNS, see *Programming an OpenLNS Application* in the *OpenLNS Programmer's Guide*. You can enable monitoring with the *Enable* method after opening a monitor set, if the *doEnable* element was set to False. You can disable monitoring using the *Disable* method. If you set the *doEnable* element to True when you invoke this method, the client application will be notified of the enabling of each monitor point in the set via the Object Server's *OnNvMonitorPointEvent* and *OnMsgMonitorPointEvent* events. This applies to both temporary and permanent monitor sets. If you create a permanent monitor set while the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**, you must change the management mode to **lcaMgmtModePropagateConfigUpdates (0)** before opening the monitor set. You can change the network management mode by writing to the value of the *MgmtMode* property of the *System* object. **Note**: You cannot open a monitor set while in engineered mode. In addition, this method does not apply to *MonitorSet* objects that were created as temporary monitor sets. All temporary *MonitorSet* objects are opened automatically as soon they are created. You can create temporary *MonitorSet* objects with the *CreateTemporaryMonitorSet* method.
Availability	Local, full, lightweight, and independent clients. Note that temporary monitor sets are not available on Independent

	clients.
Syntax	*monSetObject.Open doEnable, doPoll* **Element** — **Description** *monSetObject* — The *MonitorSet* object to be closed. *doEnable* — A Boolean value. **TRUE**. Automatically invokes the *Enable* method on the monitor set as it is opened. This enables monitoring on all of the network variable and message monitor points in the monitor set that have not been explicitly disabled with the *Disable* method. If you set the *doEnable* element to True when you invoke this method, the client application will be notified of the enabling of each monitor point in the set via the Object Server's *OnNvMonitorPointEvent* and *OnMsgMonitorPointEvent* events. This applies to both temporary and permanent monitor sets **FALSE**. If this element was set to False, you can enable monitoring with the *Enable* method after opening a monitor set. You can use the *Disable* method to turn off monitoring. *doPoll* — A Boolean value. If the *doEnable* property is set to False, this value is ignored. **TRUE**. Use polled monitoring to monitor the monitor points in the monitor set. You can set the rate at which the monitor points in the set will be polled by writing to the *PollInterval* property of the *NvMonitorOptions* object accessed through the monitor set's *NvOptions* property. **FALSE**. Use bound monitoring to

	monitor the monitor points in the monitor set.
Added to API	LNS Release 3.0.

Properties

The *MonitorSet* object contains the following properties:

- *ClassId*
- *IsEnabled*
- *IsOpen*
- *IsPollingEnabled*
- *MsgMonitorPoints*
- *MsgOptions*
- *Name*
- *NvMonitorPoints*
- *NvOptions*
- *Parent*
- *Tag*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *MonitorSet* object in the *ConstClassIds* constant: **74 lcaClassIdMonitorSet** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

IsEnabled

Summary	Indicates whether the monitor set is enabled. A monitor point must be enabled before polling, implicit bound monitoring, or explicit bound monitoring and control of the monitor point can be perfromed. You can enable all the monitor points in a monitor set at once using the *MonitorSet* object's *Enable* method. Alternatively, you can enable an individual network variable monitor point using the *NvMonitorPoint* object's *Enable* method. Message monitor points cannot be individually enabled. You can disable monitoring of a monitor set or a network variable monitor point with the *Disable* ethod. If a network variable monitor point has been explicitly disabled with the *Disable* method, it will not be re-enabled when the *Enable* method is called on the monitor set. You can only re-enable that monitor point by calling the *Enable* method on it. Similarly, if you call the *Enable* method on an *NvMonitorPoint* object, it will enable monitoring for that point only if monitoring has already been enabled for the entire monitor set.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*isEnabledFlag* = *msObject*.**IsEnabled** **Element** — **Description** *isEnabledFlag* — A Boolean value indicating whether the monitor set is currently enabled. **TRUE**. The monitor set has been enabled. **FALSE**. The monitor set has not been enabled. *msObject* — The *MonitorSet* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

IsOpen

Summary	Indicates whether the specified object is currently open. You can open the *MonitorSet* object with the *Open* method, and you can close it with the *Close* method. When you create a *MonitorSet* as a temporary monitor set, OpenLNS automatically opens it. Once a temporary monitor set is closed, it will be deleted
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*isOpenFlag* = *Object*.**IsOpe** **Element** — **Description** *isOpenFlag* — Boolean value. *Object* — A Boolean value indicating whether the *MonitorSet* object is currently open. **TRUE**. The *MonitorSet* object is currently open. **FALSE**. The *MonitorSet* object is currently closed.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

IsPollingEnabled

Summary	Indicates whether polled monitoring is enabled for the monitor set. When polling is enabled for a monitor point, OpenLNS will periodically read the value of the monitor point and report the value using the *OnNvMonitorPointEvent*. This is most efficient when the value must be checked regularly, but the application does not need to know immediately if the value changes (for example, outside air temperature). You can automatically start polled monitoring of a monitor set by setting the *doEnable* and *doPoll* elements to True when you open the set with the *Open* method. You can also start polled monitoring of a monitor set by setting the *doPoll* element to true when you enable the monitor set with the *Enable* method.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*isPollingEnabledFlag* = *msObject*.**IsPollingEnabled**

	Element	**Description**
	isPollingEnabledFlag	A Boolean value indicating whether polled monitoring is enabled. **TRUE**. Polled monitoring is enabled. **FALSE**. Polled monitoring is not enabled.
	msObject	The *MonitorSet* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

MsgMonitorPoints

Summary	Contains the collection of all message tag monitor points (*MsgMonitorPoint* objects) that have been added to the monitor set. See the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide* for more information on message monitor points.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*msgMonSetColl* = *monSetObject*.**MsgMonitorPoints** **Element** — **Description** *msgMonSetColl* — The *MsgMonitorPoints* collection object returned. *monSetObject* — The *MonitorSet* object to be acted on.
Data Type	*MsgMonitorPoints* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

MsgOptions

Summary	Contains the default monitoring options that are initially applied to the *MsgMonitorPoint* objects in this monitor set. This property stores the monitor set's *MsgMonitorOptions* object. This contains the default monitoring options that are applied to all *MsgMonitorPoint* objects as they are added to the monitor set. These options will be used to monitor these points when they are enabled.

	The *MsgMonitorOptions* object contained within this property is not passed by reference. If you acquire a *MsgMonitorOptions* object through the *MsgOptions* property and modify it, you must then explicitly assign the modified *MsgMonitorOptions* object back to the *MsgOptions* property for the changes to take effect. This following code sample demonstrates this procedure:
	`Set msgOptions = monSet.MsgOptions`
	`msgOptions.Name = "Device 003"`
	`Set monSet.MsgOptions = msgOptions`
	You should note that each *MsgMonitorPoint* object in a monitor set contains a *CurrentOptions* property and a *CurrentOptions* property. Once the monitor set containing a *MsgMonitorPoint* has been opened, you can use the monitor point's *CurrentOptions* property to set the monitoring options that will be used for the *MsgMonitorPoint* during that particular session.
	You can use a permanent message monitor point's *DefaultOptions* property to change the default monitoring options that will be applied to that particular message monitor point in future monitoring sessions. This may be useful if you want a message monitor point in a permanent monitor set to use a different set of default monitoring options than those defined for the entire monitor set. The *DefaultOptions* properties of message monitor points in temporary monitor sets are not accessible because temporary monitor sets are only used in a single client session.

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*msgOptions* = *monSetObject*.**MsgOptions** **Element** — **Description** *msgMonSetColl* — The *MsgMonitorPoints* collection object returned. *monSetObject* — The *MonitorSet* object to be acted on.
Data Type	*MsgMonitorOptions* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period

	(.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NvMonitorPoints

Summary	Contains the collection of all network variable monitor points (*NvMonitorPoint* objects) that have been added to the monitor set. See the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide* for more information on message monitor points.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*nvMonSetColl* = *monSetObject*.**NvMonitorPoints** **Element** — **Description** *nvMonSetColl* — The *NvMonitorPoints* collection object returned. *monSetObject* — The *MonitorSet* object to be acted on.
Data Type	*MsgMonitorPoints* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

NvOptions

Summary	Contains the default monitoring options that are initially applied to the *NvMonitorPoint* objects in the monitor set. This property stores the monitor set's *NvMonitorOptions* object. This contains the default monitoring options that are applied to all *NvMonitorPoint* objects as they are added to the monitor set. These options will be used to monitor these points when they are enabled.

	The *NvMonitorOptions* object contained within this property is not passed by reference. If you acquire a *NvMonitorOptions* object through the *NvOptions* property and modify it, you must then explicitly assign the modified *NvMonitorOptions* object back to the *NvOptions* property for the changes to take effect. This following code sample demonstrates this procedure:
	`Set nvOptions = monSet.NvOptions`
	`nvOptions.Name = "Device 003"`
	`Set monSet.NvOptions = nvOptions`
	Each *NvMonitorPoint* object in a monitor set contains a *CurrentOptions* property and a *DefaultOptions* property. Once the monitor set containing an *NvMonitorPoint* has been opened, you can use the monitor point's *CurrentOptions* property to set the monitoring options that will be used for the *NvMonitorPoint* during that particular session.
	You can use a permanent network variable monitor point's *DefaultOptions* property to change the default monitoring options that will be applied to that particular network variable monitor point in future monitoring sessions. This may be useful if you want a network variable monitor point in a permanent monitor set to use a different set of default monitoring options than those defined for the entire monitor set. The *DefaultOptions* properties of network variable monitor points in temporary monitor sets are not accessible because temporary monitor sets are only used in a single client session.

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*nvOptions* = *monSetObject*.**NvOptions** **Element** — **Description** *nvOptions* — The *NvMonitorOptions* object containing the default network variable monitoring options. *monSetObject* — The *MonitorSet* object to be acted on.
Data Type	*NvMonitorOptions* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent

	clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Tag

Summary	Stores any extra data associated with the monitor point or monitor set. The data stored in this property is not used by OpenLNS, and is available as soon as the data point or monitor set is created in a monitor and control session. You can use this property to store any data your application may need when using the monitor set or monitor point. For example, you could store the name of the monitor set that a message or network variable monitor point belongs to, or the name of the application device that a monitor set belongs to. The *Tag* properties for all *NvMonitorPoint* and *MsgMonitorPoint* objects in permanent monitor sets are cached when the monitor set is opened. As a result, any changes made the *Tag* properties of these monitor points while the permanent monitor set is open will not be accessible until the monitor set is closed and re-opened. When initially created, monitor points and monitor sets will have a null Tag value. However, if you add a monitor point to an open monitor set and set its *Tag* value in the same transaction, you will be able to access the *Tag* value during that monitor and control session, as all data would be written to the device as soon as the transaction is committed. You should note that this behavior does not apply to monitor points in temporary monitor sets. Temporary monitor sets support "live" updates to the value of the *Tag* property. A well-defined monitoring application will include any information necessary to quickly and efficiently identify the monitor point in this property. This will eliminate the need to gather such information from the database, or to perform other time-consuming activities, during the monitoring process.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on

	Independent clients.
Syntax	*tagValue* = *Object*.**Tag** **Element** — **Description** *Object* — The monitor point or monitor set object to be acted on. *tagValue* — The tag associated with the object.
Data Type	*Variant*.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

MonitorSets

The *MonitorSets* object contains a collection of *MonitorSet* objects representing all the monitor sets on a device. If a device does not support monitor sets, an empty collection will be returned when you attempt to access this collection. In general, only NetworkServiceDevices (NSDs) support monitor sets. The following table summarizes the *MonitorSets* object.

Description	A collection of *MonitorSet* objects.
Added to API	LNS Release 3.0.
Accessed Through	*AppDevice* object. *Network* object.
Default Property	*Item*
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *MonitorSets* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	Adds a permanent *MonitorSet* object to the collection. This method can only be called when connected to the OpenLNS Server (after the *Network* object's *Open* method has been called). You should not add and *open* a monitor set in the same explicit transaction. If you create a monitor set while the *network management mode* is set to

	lcaMgmtModeDeferConfigUpdates (1), you must change the management mode to **lcaMgmtModePropagateConfigUpdates (0)** before opening the monitor set. You can do so by setting the *MgmtMode* property of the *System* object to **lcaMgmtModePropagateConfigUpdates (0)**. You can access the *MonitorSets* collection through the *MyVNI* and *CurrentMonitorSets* properties of the *Network* object. You should use the *MyVNI* property when you are going to add new *MonitorSet* objects to the collection. For more information, see the *MonitorSet* object. **Note:** You can create *MonitorSet* objects while running in engineered mode, but you cannot open them. If you attempt to do so, the **LCA, #143 lcaErrNotAllowedWithoutNetworkInterface** exceptions will be thrown.
Availability	Local, full, lightweight, and independent clients.
Syntax	*monSetObject* = *monSetColl*.**Add** *monSetName* **Element** — **Description** *monSetObject* — The newly defined *MonitorSet* object. *monSetColl* — The *MonitorSets* collection object to be acted upon. *monSetName* — The *Name* of the new *MonitorSet* object.
Added to API	LNS Release 3.0.

Remove

Summary	Removes a *MonitorSet* object from the collection. This method can only be called when connected to the network database (after the *Network* object's *Open* method has been called). This method only affects *MonitorSet* objects that were created as permanent *MonitorSet* objects becuase OpenLNS deletes temporary *MonitorSet* objects automatically as soon as the client session in which they were created ends. See the *MonitorSet* object for more information on the differences between temporary and permanent *MonitorSet* objects.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects, and temporary monitor points, are not available on Independent clients.
Syntax	*monSetColl*.**Remove** *indexName* **Element** — **Description** *monSetColl* — The *MonitorSets* collection object to be acted upon. *indexName* — A Long value specifying the collection

	index of the *MonitorSet* object to be removed, or a String value specifying the name of the *MonitorSet* object to be removed.
Added to API	LNS Release 3.0.

Properties

The *MonitorSets* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *MonitorSets* object in the *ConstClassIds* constant: **75 lcaClassIdMonitorSets** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count**

	Element	**Description**
	returnValue	The number of objects in the collection as a long integer.
	object	The collection object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Item

Summary	Returns a *MonitorSet* object from a *MonitorSets* collection. You can retrieve a *MonitorSet* object from its *MonitorSets* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *MonitorSet* object in *MonitorSets* collections with the *Name* property by passing the object's name as a string expression.	
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.	
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*)	
	Element	**Description**
	retrievedObject	The *MonitorSet* object retrieved from the collection.
	collObject	The *MonitorSets* collection object to be acted on.
	index	A Long type specifying the ordinal index of the *MonitorSet* object to be retrieved.
	stringExpression	A string type specifying the name of the *MonitorSet* object to be retrieved.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.

Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

MsgMonitorOptions

The *MsgMonitorOptions* object contains a set of monitoring options that can be applied to the message monitor points that OpenLNS is monitoring. Depending on how this object is accessed, writing to this it may affect the monitoring options used for an entire monitor set, or for an individual message tag monitor point.

The *MsgMonitorOptions* object accessed through the *MsgOptions* property of a *MonitorSet* object contains the default monitoring options for the monitor set. These are the default options applied to all *MsgMonitorPoint* objects as they are added to the monitor set.

You can change the defaults an individual *MsgMonitorPoint* uses by writing to the *MsgMonitorOptions* object accessed through the *DefaultOptions* property of that *MsgMonitorPoint* object. This object contains the default options that will be used each time the monitor set containing that particular *MsgMonitorPoint* object is opened. Note that this only applies to message monitor points in permanent monitor sets. The *CurrentOptions* properties of message monitor points in temporary monitor sets are not accessible, as temporary monitor sets are only used in a singe client session.

You can change the monitoring options a *MsgMonitorPoint* object will use for an active monitoring session by writing to the *MsgMonitorOptions* object accessed through the *MsgMonitorPoint* object's *CurrentOptions* property. Changes made to the current options take effect for the current session only, and are not stored persistently in memory.

Note that the *MsgMonitorOptions* object contained within the *MsgOptions* property of a monitor set (or the *CurrentOptions* properties of a monitor point) is not passed by reference. If you acquire a *MsgMonitorOptions* object through any of these properties and modify it, you must then explicitly assign the modified object back to the source property for the changes to take effect.

The following table summarizes the *MsgMonitorOptions* object.

Description	A set of monitoring options that can be applied to the message monitor points being monitored by OpenLNS.
Added to API	LNS Release 3.0.
Accessed Through	*MonitorSets* object. *MsgMonitorPoint* object.
Default Property	None.
Methods	None.
Properties	• *Authentication* • *ClassId* • *FilterByCode* • *FilterBySource* • *FilterCode* • *Parent* • *Priority* • *Retries* • *ServiceType*

	• *UseAsyncSend*

Methods

The *MsgMonitorOptions* object does not contain any methods.

Properties

The *MsgMonitorOptions* object contains the following properties:

- *Authentication*
- *ClassId*
- *FilterByCode*
- *FilterBySource*
- *FilterCode*
- *Parent*
- *Priority*
- *Retries*
- *ServiceType*
- *UseAsyncSend*

Authentication

Summary	Determines whether the authenticated service will be used when sending a message monitor point updates using the *Value* property. See the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide* for more information on message monitor points.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*authValue* = *monOptsObject*.**Authentication** **Element** — **Description** *authValue* — Boolean value. *monOptsObject* — The *MsgMonitorOptions* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to

	Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *MsgMonitorOptions* object in the *ConstClassIds* constant: **76 lcaClassIdMsgMonitorOptions** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

FilterByCode

Summary	Indicates whether the *FilterCode* property will be used to filter message tag values.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*filterByCode* =*msgMonOpts*.**FilterByCode** **Element** **Description** *filterByCode* **TRUE**. Message tag values are filtered by the *FilterCode* property before being passed to the client. **FALSE**. The *FilterCode* property is ignored. *msgMonOpts* The *MsgMonitorOptions* object.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

FilterBySource

Summary	Indicates whether message tag values will be filtered by the source device. This property can be combined with the *FilterByCode* property to allow only message tags with certain codes from certain sources to be passed to your client application via a *MsgMonitorPoint* object.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*filterBySource* =*msgMonOpts*.**FilterBySource** **Element** — **Description** *filterBySource* — A Boolean value. **TRUE**. Message tag values are filtered by the target *AppDevice* object that was specified when the monitored message tag point was created (using the *Add* method). **FALSE**. The message tag source is ignored. *msgMonOpts* — The *MsgMonitorOptions* object.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

FilterCode

Summary	Contains the filter code that will be used if the *FilterByCode* property is set to True. Each explicit message contains a 1-byte message code along with the message data. If the *FilterByCode* property is set to True and this property is set, only messages with a message code matching the value of this property will be passed to the client via the *MsgMonitorPoint* objects using this *MsgMonitorOptions* obect.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*filterCode* =*msgMonOpts.FilterCode* **Element** — **Description** *filterCode* — The filter code. This element may be set to a value between 0–255; however, messages with

		codes in the range 80–126 and 128–255 cannot be received by an OpenLNS application. This is because these codes are used for network management, diagnostic, and network variable messages, and they are processed directly by the NSD.
	msgMonOpts	The *MsgMonitorOptions* object.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Priority

Summary	Determines whether priority is used when polling or explicitly updating a message monitor point using the *Value* property. See the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide* for more information
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*priorityValue* = *monOptsObject*.**Priority** **Element** — **Description**

	priorityValue	A Boolean value. **TRUE**. Priority is used when polling or explicitly updating a message monitor point using the *Value* property. **FALSE**. Priority is not used.
	monOptsObject	The *MsgMonitorOptions* to be acted on.
Data Type	Boolean.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

Retries

Summary	Specifies the number of retries to use for acknowledged, request/response, or repeated service messages. This applies to network management messages only. Setting this property through the *NvOptions* or *MsgOptions* properties of a permananent monitor set, or through the *CurrentOptions* property of a permanent monitor point, does not have an effect. The default retry count to use for all permanent network variable and message monitor points is established by the *RetryCount* property of the *ConnectDescTemplate* specified when the monitor point was created. You cannot set this property through the *DefaultOptions* property of a temporary monitor point, as the *CurrentOptions* properties of all temporary monitor points are not accessible. However, you can set this property through the *NvOptions* or *MsgOptions* property of a temporary monitor set to determine the default retry count that will be applied to all temporary monitor points as they are added to the set. You can also set this property through the *CurrentOptions* property of a permanent or temporary monitor point to determine what retry count for the current monitoring session. See the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide* for more information.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*retryValue* = *monOptsObject*.**Retries** **Element** — **Description** *retryValue* — The number of retries. The valid range of values for this property is

		1–15. This property applies to network management messages only.
	monOptsObject	The *MsgMonitorOptions* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

ServiceType

Summary	Determines the service type used when sending explicit messages with a message monitor point. You cannot set this property through the *CurrentOptions* property of a temporary monitor point, as the *CurrentOptions* properties of all temporary monitor points are not accessible. However, you can set this property through the *NvOptions* or *MsgOptions* property of a temporary monitor set to determine the default messaging service that will be applied to all monitor points as they are added to the set. See the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide* for more information.	
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.	
Syntax	*retryValue* = *monOptsObject*.**Retries**	
	Element	**Description**
	retryValue	The valid values for this property, which are enumerated in the *ConstServiceTypes* constant, are as follows: **0 lcaSvcAckd** Acknowledged messaging service. The device sends an acknowledgment message after it has received the message. If your application will be sending messages to large numbers of devices at once, one of the unacknowledged messaging services may be desirable, as the acknowledgment messages may generate a significant amount of network traffic. **1 lcaSvcUnackdRpt** Unacknowledged repeat messaging

		service. The device does not send acknowledgment messages; however, repeat messages are sent to the device after the initial message is sent to it to ensure that it reaches its destination. You can set the number of repeat messages to send, and the interval at which they will be sent, by writing to the *RepeatCount* and *RepeatTimer* properties. **2 lcaSvcUnackd** Unacknowledged messaging service. The device does not send acknowledgment messages. Do not use this service type on channels that support altnerate frequencies because the message will only be sent using the primary path. See the *AltPathType* property for more information. **3 lcaSvcRequest** Request/Response messaging service. You can use this value when sending explicit messages if the device receiving the message is designed to send a response message for the specified message code.
	monOptsObject	The *MsgMonitorOptions* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

UseAsyncSend

Summary	Determines whether OpenLNS will wait for a completion code to return after updating the value of a monitor point before sending its next update message. When sending the values of network variable and message monitor points prior to LNS 3.20, LNS would wait for the completion code for each message sent to be returned before returning to the user, regardless of the messaging service type being used. You can use this property to determine whether OpenLNS should wait for the completion code. **Note:** The *UseAsyncSend* property can be used to confirm when the values of the monitor points on your network have

<table>
<tr><td></td><td>been successfully sent, as described earlier. However, you can also use the OnMsgMonitorPointErrorEvent and OnNvMonitorPointErrorEvent events to determine when values are not successfully sent. These events are generated whenever there is a write failure on a monitor point.

See the Monitor and Control chapter in the OpenLNS Programmer's Guide for more information.</td></tr>
<tr><td>Availability</td><td>Local, full, lightweight, and independent clients. Note that temporary MonitorSet objects and temporary monitor points are not available on Independent clients.</td></tr>
<tr><td>Syntax</td><td>monOptsObject.UseAsyncSend = asyncFlag

Element — Description

monOptsObject — The MsgMonitorOptions object being acted upon.

asyncFlag — A Boolean value indicating whether OpenLNS waits for a completion code to return after updating the value of the monitor point before sending its next update message.

TRUE. OpenLNS does not wait for the completion code to return after sending the values of the monitor points before returning to the user.

In this case, OpenLNS will generate an OnMsgMonitorPointEvent or OnNvMonitorPointEvent event as soon as the completion code has been returned, and the value of the monitor point has been updated. You can use these events to confirm that the values of your monitor points have been successfully sent.

This approach may be useful if you are updating a large number of monitor points at once, and do not want to wait for a completion code to return after each update before moving to the next one.

Setting this property to True therefore may be useful when writing to the values of large numbers of data points</td></tr>
</table>

	with the *Write* method because it reduces the time required to update the values on the network. **FALSE**. OpenLNS waits for the completion code to return after it sends each value before returning from the method, as with LNS versions prior to LNS 3.20. Other client applications will be able to successfully update the monitor point while your application is waiting for the completion code. If this property is set to False and no completion code is returned after a message is sent, this indicates that OpenLNS failed to update the monitor point. In this case, the **DS, #411 lcaErrLnsDsWriteFailed** exception will be thrown before OpenLNS proceeds to the next update message. OpenLNS will not return any events confirming that the value has been updated when the property is set to False. This is the default.
Data Type	Boolean.
Read/Write	Read/write. **Note**: OpenLNS sets this property automatically for message monitor points. For example, if you are sending a message via the *SendMsgWait* method, a response from the device is expected. Therefore, OpenLNS will set the *UseAsyncSend* property to False. Also, if you are writing to a message monitor point via the *SendMsgWait* property, no response is expected; therefore, OpenLNS sets the property to True. As a result, you should not write to this property when you access it through a *MsgMonitorOptions* object.
Added to API	LNS Release 3.20.

MsgMonitorPoint

A *MsgMonitorPoint* represents a single monitored or controlled message tag. You can use message monitor points to monitor LONWORKS messages from application devices on your network with your application. For an overview of message monitor points and how they work, see the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide*.

In general, the *MsgMonitorPoint* objects contained in *MonitorSet* objects created as temporary or permanent monitor sets behave the same way except that the *DefaultOptions* properties of the *MsgMonitorPoint* objects in temporary monitor sets are not accessible. For more information on this, see the *CurrentOptions* property of this object.

Temporary monitor sets, and all the monitor points they contain, can only be used in a single client session. They are deleted by OpenLNS as soon as the session in which they were created ends. For more information on the differences between temporary and permanent monitor sets, see the *MonitorSet* object.

The following table summarizes the *MsgMonitorPoint* object.

Description	A single monitored or controlled message tag.
Added to API	LNS Release 3.0.
Accessed Through	*MsgMonitorPoints* collection object.
Default Property	*Name*.
Methods	• *Advise* • *Disable* • *Enable* • *SendMsgWait* • *Unadvise*
Properties	• *ClassId* • *CurrentOptions* • *DefaultOptions* • *InputFormatSpec* • *Name* • *OutputDataPoint* • *OutputFormatSpec* • *Parent* • *RequestDataPoint* • *Tag*
Events	• *UpdateErrorEvent* • *UpdateEvent*

Methods

The *MsgMonitorPoint* contains the following methods.

- *Advise*
- *Disable*
- *Enable*
- *SendMsgWait*
- *Unadvise*

Advise

Summary	Enables update and error events for an object that implements the *ILcaMsgMonitorPointListener* or *ILcaNvMonitorPointListener* interface. This method should only be used if the development environment supports multi-threading (such as Visual C++). When you call this method, OpenLNS will provide event notification of updates and update errors using callbacks, instead of Windows messaging. The callback is made to the *UpdateEvent* or *UpdateErrorEvent* method of the object specified as the *object* parameter when the *Advise* method is called. When you call the *Advise* method, the client thread will stop generating *OnMsgMonitorPointErrorEvent* and *OnMsgMonitorPointUpdateEvent* events for the *MsgMonitorPoint* object specified as the *mpObject* element. The object specified as the *object* element will then start receiving *UpdateErrorEvent* events and message *UpdateEvent* events for that monitor point. **Note**: The *Advise* method must be called from the event handler that is managing the *MsgMonitorPoint* events listed above. You should determine how these updates are handled by defining the *UpdateErrorEvent* method and the *UpdateEvent* method for the *ILcaMsgMonitorPointListener* interface. The returned tag should be supplied to the *Unadvise* method to return update notification to the client thread.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*tag* = *mpObject*.**Advise** *object* **Element** — **Description** *tag* — An Integer type *mpObject* — The *MsgMonitorPoint* object to be acted on. *object* — An object that implements the *ILcaMsgMonitorPointListener* interface.
Added to API	LNS Release 3.0.

Disable

Summary	Disables monitoring of a message monitor point. When you disable monitoring of a message monitor point, you should note that this overrides subsequent calls to the *MonitorSet* object's *Enable* method.

	For example, if you call the *Disable* method on a *MsgMonitorPoint* object named Point A, and then call the *Disable* method on the monitor set containing Point A, Point A would not be enabled. All other monitor points in the monitor set, however, would be enabled. Once you have explicitly disabled a *MsgMonitorPoint* with the *Disable* method, you can only re-enable that monitor point by calling the *Enable* method on it, or by closing and re-opening the monitor set it belongs to. You can also disable an entire monitor set by calling the *Disable* method on the *MonitorSet* object. When you do this, polled and bound monitoring for all monitor points on the monitor set will be disabled. After this, none of the monitor points in the set can be enabled for monitoring until the *Disable* method has been called on the *MonitorSet* object again. For more details on opening and enabling monior sets and monitor points, see the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*object.Disable* **Element** — **Description** *object* — The *MsgMonitorPoint* object to be disabled.
Added to API	LNS Release 3.20.

Enable

Summary	Enables monitoring of a message monitor point. You can enable monitoring of all the monitor points in a permanent or temporary monitor set at once by setting the *doEnable* element to True when you *open* the monitor set. If the *doEnable* element is set to False, you can also enable the entire monitor set later by calling the *Enable* method on the *MonitorSet* object. You can disable monitoring of an individual message monitor point by calling the *Disable* method on it. For more details on opening and enabling monior sets and monitor points, see the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.

Syntax	*object*.**Ensable** **Element** **Description** *object* The *MsgMonitorPoint* object to be enabled.
Added to API	LNS Release 3.20.

SendMsgWait

Summary	Sends a message that requires a synchronous response on a *MsgMonitorPoint* object created using the *GetMessagePoint* method. The *AppDevice* object's *GetMessagePoint* method creates an *MsgMonitorPoint* object which can be used to send messages to the device. The *MsgMonitorPoint* object's *RequestDataPoint* property can be used to send a request response message to the device. To send the message to the device, set the *RequestDataPoint* object's *Value*, *RawValue* , or *FormattedValue* property then call this method or the *Write* method. If this method is used, the request and response datapoints will be sent in the same call (synchronously). If the *Write* method is used, the response will be sent as an *OnMsgMonitorPointUpdateEvent* event.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*responseDpObject* = *msgMpObject.SendMsgWait*(*requestDpObject)* **Element** **Description** *responseDpObject* The *DataPoint* object generated by the device in response to the *requestDpObject*. *msgMpObject* The *MsgMonitorPoint* object to be acted on. *requestDpObject* The *DataPoint* object to be sent to the device.
Added to API	LNS Release 3.0.

Unadvise

Summary	Returns event generation to the client thread if it was changed to another thread using the *Advise* method.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.

Syntax	*mpObject*.**Unadvise** *tag* **Element** **Description** *tag* An Integer type. This *tag* parameter should use the tag that was returned when the *Advise* method was called. *mpObject* The *MsgMonitorPoint* object to be acted on.
Added to API	LNS Release 3.0.

Properties

The *MsgMonitorPoint* object contains the following properties:

- *ClassId*
- *CurrentOptions*
- *DefaultOptions*
- *InputFormatSpec*
- *Name*
- *SendMsgWait*
- *OutputFormatSpec*
- *Parent*
- *RequestDataPoint*
- *Tag*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *MsgMonitorPoint* object in the *ConstClassIds* constant: **77 lcaClassIdMsgMonitorPoint** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CurrentOptions

Summary	Contains the current monitoring options for this message monitor point. Once the monitor set containing a monitor point has been opened, you can use this property to set the monitoring options that will be used for the monitor point during that particular session.
	
	For monitor points in *MonitorSet* objects created as permanent monitor sets, the options contained in this property default to the options set in the *DefaultOptions* property of the *MsgMonitorPoint*. If the options are not set there, they default to the options set in the *MsgOptions* and *NvOptions* properties in the *MonitorSet* object.
	
	Each time a permanent *MonitorSet* object's *Open* method is called, the current options for each of the monitor points in the set are reset to the options contained in their *CurrentOptions* properties. The *CurrentOptions* property can only be written to when the monitor set is open.
	
	The *MsgMonitorOptions* object contained within this property is not passed by reference. If you acquire a *MsgMonitorOptions* object through the *CurrentOptions* property and modify it, you must then explicitly assign the modified object back to the *CurrentOptions* property for the changes to take effect. This following code sample demonstrates this procedure:
	
	`Set curOptions = monPoint.CurrentOptions`
	`curOptions.Authentication = True`
	`curOptions.Retries = 5`
	`Set monPoint.CurrentOptions = curOptions`

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*curOptions* = *monPoint.CurrentOptions* **Element** — **Description** *curOptions* — The *MsgMonitorOptions* object containing the current options for this monitor point. *monPoint* — The *MsgMonitorPoint* object to be acted on.
Data Type	*MsgMonitorOptions* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

DefaultOptions

Summary	Contains the default monitoring options that are applied each time the monitor set containing this network variable or

<table>
<tr><td></td><td>message monitor point is opened.

These options are read when the monitor set containing the monitor point is opened. Changes to these options will not take effect until the next time the monitor set is opened. Use the CurrentOptions property to change the active monitoring options to use for a monitor point that is currently enabled.

For message monitor points, the options contained in the DefaultOptions property default to the options set in the permanent monitor set's MsgOptions property.

The default options cannot be accessed in server-independent mode; therefore, Independent clients cannot read or write to the DefaultOptions property.

The MsgMonitorOptions object contained within this property is not passed by reference. If you acquire a MsgMonitorOptions object through the DefaultOptions property and modify it, you must then explicitly assign the modified object back to the DefaultOptions property for the changes to take effect. This following code sample demonstrates this procedure:

<pre>Set defOptions = monPoint.DefaultOptions
defOptions.Authentication = True
Set monPoint.DefaultOptions = defOptions</pre></td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>defOptions = monPoint.CurrentOptions

Element — Description
defOptions — The MsgMonitorOptions object containing the default options for this monitor point.
monPoint — The MsgMonitorPoint object to be acted on.</td></tr>
<tr><td>Data Type</td><td>MsgMonitorOptions object.</td></tr>
<tr><td>Read/Write</td><td>Read/write.

Note: The DefaultOptions properties of monitor points in MonitorSet objects created as temporary monitor sets are not accessible. If you attempt to acquire the DefaultOptions property through a temporary monitor point, the LCA, #161 lcaErrNotAllowedOnTemporaryObject exception is thrown. This is because these monitor points can only be used in a singe client session.

If you want a temporary monitor point to use options other than the defaults for the monitor set, you can change them with the DefaultOptions property.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.0.</td></tr>
</table>

InputFormatSpec

Summary	Contains the *FormatSpec* object used to determine the format for incoming messages for this *MsgMonitorPoint* object.
	
	The *FormatSpec* object contained within this property is not passed by reference. If you modify the values assigned to the properties of a local *FormatSpec* object, you must then explicitly assign the modified *FormatSpec* object back to the *InputFormatSpec* property of the *MsgMonitorPoint* for the changes to take effect. This following code sample demonstrates this procedure:
	
	`Set fsObject = msgMonitorPointObject.InputFormatSpec`
	
	`fsObject.AltFormatName = "SNVT_temp_f#SI"`
	
	`Set msgMonitorPointObject.InputFormatSpec = fsObject`

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*fsObject* = *msgMpObject*.**InputFormatSpec** **Element** — **Description** *fsObject* — The *FormatSpec* object used to interpret incoming messages. *msgMpObject* — The *MsgMonitorPoint* object to be acted on.
Data Type	*FormatSpec*.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.

Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

OutputDataPoint

Summary	Contains an *OutputDataPoint* object that can be used to send a message to the device monitored by the *MsgMonitorPoint* object, as long as the message does not require a response. When the *AppDevice* object's *GetMessagePoint* method is called, the *DataPoint* object that is created can read this property to send a non-request message to that device. The *DataPoint* contained in this property has its *AutoWrite* property set to True by default; therefore, you do not need to call the *Write* method after setting this *DataPoint* object's *Value* property. The monitor set containing the *MsgMonitorPoint* must be open in order for you to access this data point Use the *RequestDataPoint* property to send a message that requires a response.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*dpObject = msgMpObject.OutputDataPoint* **Element** — **Description** *dpObject* — The *DataPoint* object to be returned. *msgMpObject* — The *MsgMonitorPoint* object to be acted on.
Data Type	*DataPoint* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

OutputFromatSpec

Summary	Contains the *FormatSpec* object used to determine the format for outgoing messages for this *MsgMonitorPoint* object. The *FormatSpec* object contained within this property is not passed by reference. If you modify the values assigned to the properties of a local *FormatSpec* object, you must then explicitly assign the modified *FormatSpec* object back to the *OutputFormatSpec* property of the *MsgMonitorPoint* for the changes to take effect. This following code sample demonstrates this procedure: `Set fsObject =`

	`msgMonitorPointObject.OutputFormatSpec`
	`fsObject.AltFormatName = "SNVT_temp_f#SI"`
	`Set msgMonitorPointObject.OutputFormatSpec = fsObject`

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*fsObject* = *msgMpObject*.**OutputFormatSpec** **Element** — **Description** *fsObject* — The *FormatSpec* object used to format outgoing messages. *msgMpObject* — The *MsgMonitorPoint* object to be acted on.
Data Type	*FormatSpec* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

RequestDataPoint

Summary	Returns a *DataPoint* object that you can use to send a request message to the device monitored by this *MsgMonitorPoint* object. The device should send a response to this message. The monitor set containing the *MsgMonitorPoint* must be

<table>
<tr><td></td><td>open in order for you to access this property. When the AppDevice object's GetMessagePoint method is called, the DataPoint object that is created can set this property to send a request message to the device monitored by this MsgMonitorPoint object.

The AutoWrite property of this DataPoint object is set to False; therefore, you need to explicitly propagate the message to the network after the DataPoint object's Value, RawValue , or FormattedValue property is set. You can do this in two ways:

• Call the DataPoint object's Write method to receive a response via the OnMsgMonitorPointUpdateEvent event (i.e. asynchronous response).

• Call the MsgMonitorPoint object's SendMsgWait method with this DataPoint as the object to receive a response as the return of the SendMsgWait method (i.e. synchronus response).

You can use the OutputDataPoint property to send a message that does not require a response.</td></tr>
<tr><td>Availability</td><td>Local, full, lightweight, and independent clients. Note that temporary MonitorSet objects and temporary monitor points are not available on Independent clients.</td></tr>
<tr><td>Syntax</td><td>dpObject = msgMpObject.RequestDataPoint

Element — Description
dpObject — The DataPoint object returned.
msgMpObject — The MsgMonitorPoint object to be acted on.</td></tr>
<tr><td>Data Type</td><td>DataPoint object.</td></tr>
<tr><td>Read/Write</td><td>Read/write.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.0.</td></tr>
</table>

Tag

Summary	Stores any extra data associated with the monitor point or monitor set. The data stored in this property is not used by OpenLNS, and is available as soon as the data point or monitor set is created in a monitor and control session. You can use this property to store any data your application may need when using the monitor set or monitor point. For example, you could store the name of the monitor set that a message or network variable monitor point belongs to, or the name of the application device that a monitor set belongs to. The *Tag* properties for all *MsgMonitorPoint* objects in permanent monitor sets are cached when the monitor set is opened. As a result, any changes made the *Tag* properties of these monitor points while the permanent monitor set is open will not be accessible until the monitor set is closed and re-opened. When initially created, monitor points and monitor sets will have a null Tag value. However, if you add a monitor point to an open monitor set and set its *Tag* value in the same transaction, you will be able to access the *Tag* value during that monitor and control session, as all data would be written to the device as soon as the transaction is committed. You should note that this behavior does not apply to monitor points in temporary monitor sets. Temporary monitor sets support "live" updates to the value of the *Tag* property. A well-defined monitoring application will include any information necessary to quickly and efficiently identify the monitor point in this property. This will eliminate the need to gather such information from the database, or to perform other time-consuming activities, during the monitoring process.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*tagValue* = *Object*.**Tag** **Element** — **Description** *Object* — The monitor point or monitor set object to be acted on. *tagValue* — The tag associated with the object.
Data Type	*Variant*.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Events

The *MsgMonitorPoint* object contains the following events:

- *UpdateErrorEvent*
- *UpdateEvent*

UpdateErrorEvent

Summary	Indicates that a write failure or a poll failure has occurred on the message monitor point that generated the event. This event can only be used in development environments that support events being generated on threads other than the client thread (such as Visual C++). In these environments, it is usually more efficient to generate events from a non-client thread. To cause events to be generated in this manner, create an object which implements the *ILcaNvMonitorPointListener* or *ILcaMsgMonitorPointListener* interface and call a *MsgMonitorPoint* object's *Advise* method with the created object as the argument. The object will now receive these events directly. The behavior of the object depends on how the user implements the *UpdateEvent* method.
Syntax	**UpdateErrorEvent**(*UpdateType* as Integer) **Element** — **Description** *updateType* — This element always returns the value 1.
Data Type	Integer.
Added to API	LNS Release 3.0.

UpdateEvent

Summary	Indicates that a message monitor point update has arrived. This event can only be used in development environments that support events being generated on threads other than the client thread (such as Visual C++). In these environments, it is usually more efficient to generate events from a non-client thread. To cause events to be generated in this manner, create an object which implements the *ILcaNvMonitorPointListener* or *ILcaMsgMonitorPointListener* interface and call a *MsgMonitorPoint* object's *Advise* method with the created object as the argument. The object will now receive these events directly. The behavior of the object depends on how the user implements the *UpdateEvent* method. For completion code messages, the *InputDp*, *OutputDp*, and *Src* parameters are NULL.
Syntax	**UpdateErrorEvent**(*UpdateType* as Integer)

	Element	Description
	UpdateType	This element always returns the value 0. The possible values for this parameter, which are contained in the *ConstMonitorEventType* constant, are as follows: **0 lcaMonitorEventTypeNull** This value is not used. **1 lcaMonitorEventTypeQuit** This value is not used. **2 lcaMonitorEventTypeAdd** This value is not used. **3 lcaMonitorEventTypeRemove** This value is not used. **4 lcaMonitorEventTypeMsCreate** This value is not used. **5 lcaMonitorEventTypeMsDelete** This value is not used. **6 lcaMonitorEventTypeMsChange** This value is not used. **7 lcaMonitorEventTypeMsError** This value is not used. **8 lcaMonitorEventTypeNvCreate** A network variable monitor point has been created. **9 lcaMonitorEventTypeNvDelete** A network variable monitor point has been removed. **10 lcaMonitorEventTypeNvChange** This value is not used. **11 lcaMonitorEventTypeNvError** A network variable monitor point has returned an error. See the *OnNvMonitorPointErrorEvent*. **12 lcaMonitorEventTypeNvUpdate** A network variable monitor point has received an update. See the *OnNvMonitorPointUpdateEvent*.

		13 lcaMonitorEventTypeNvComplete A completion code has returned for the monitor point. **14 lcaMonitorEventTypeMsgCreate** A message monitor point has been created. **15 lcaMonitorEventTypeMsgDelete** A message monitor point has been removed. **16 lcaMonitorEventTypeMsgChange** This value is not used. **17 lcaMonitorEventTypeMsgError** A message monitor point has returned an error. See the *OnMsgMonitorPointErrorEvent*. **18 lcaMonitorEventTypeMsgUpdate** A message monitor point has received an update. See the *OnMsgMonitorPointUpdateEvent*. **19 lcaMonitorEventTypeMsgRequest** A message monitor point has received a request message. **20 lcaMonitorEventTypeMsgResponse** A message monitor point has received a reponse message. **21 lcaMonitorEventTypeMsgComplete** This value is not used.
	InputDp	A *DataPoint* object containing the received value. The *InputDp* parameter's *AutoRead* and *AutoWrite* properties are set to False.
	OutputDp	A *DataPoint* object that allows a response to be sent if the *UpdateType* is Request. The *OutputDp* parameter's *AutoWrite* property is set to True; therefore, it will be sent as a response when you write to this *DataPoint* object.
	Src	A *SourceAddress* object indicating the source device of the update.
Data Type	Integer.	

Added to API	LNS Release 3.0.

MsgMonitorPoints

The *MsgMonitorPoints* object contains a collection of *MsgMonitorPoint* objects. This collection object contains all of the *MsgMonitorPoint* objects that have been added to a monitor set. Message monitor point objects represent message tags that may be used to monitor application LONWORKS messages from the device. For more information on monitor points, see the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide*.

In general, the *MsgMonitorPoint* collections contained in *MonitorSet* objects created as temporary and permanent monitor sets behave the same way except that the *DefaultOptions* properties of the *MsgMonitorPoint* objects in temporary monitor sets are not accessible. For more information, see the *CurrentOptions* property of the *MsgMonitorPoint* object.

The following table summarizes the *MsgMonitorPoints* object.

Description	A collection of *MsgMonitorPoint* objects.
Added to API	LNS Release 3.0.
Accessed Through	*MonitorSet* object.
Default Property	*Item*
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *MsgMonitorPoints* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	Defines a new *MsgMonitorPoint* object. You can use message monitor points to send LONWORKS messages to the devices on your network. When creating a *MsgMonitorPoint* object in a permanent monitor set, you can specify either an *AppDevice* object or a dynamic *MessageTag* object as the *targetDevice* element. • If you specify an *AppDevice* object, you will be able to use the new *MsgMonitorPoint* object to send messages to that device, or to receive messages from the device. This cannot be the Network Service Device containing the monitor set you are adding the message point to (the *AppDevice* containing the monitor set). • If you specify a dynamic *MessageTag* object, you will be

	able to use the *MsgMonitorPoint* to send messages to all the devices that the *MessageTag* object is bound to at once. You will not be able to use the *MsgMonitorPoint* to receive messages from those devices. The dynamic *MessageTag* object must exist on the Network Service Device containing the monitor set you are adding the message point to. When you are adding *MsgMonitorPoint* objects to temporary monitor sets, you can only specify an *AppDevice* object as the *targetDevice* element. You will be able to use the new *MsgMonitorPoint* object to send messages to that device, or to receive messages from the device. As with permanent monitor sets, you cannot specify the Network Service Device containing the monitor set as the *targetDevice* element. If you are adding a *MsgMonitorPoint* object to a temporary monitor set, and specify a *MessageTag* object as the *targetDevice* element, the **LCA, #161 lcaErrNotAllowedOnTemporaryObject** exception will be thrown. **Note:** If you create a *MsgMonitorPoint* object and specify an *AppDevice* as the *targetDevice* element, and that device is removed, then the *MsgMonitorPoint* object will be deleted as well. The *FilterBySource* property of the *MsgMonitorOptions* object must be set to True to restrict the *MsgMonitorPoint* object to monitoring messages from the device set specified by the *targetDevice* element. You should use transactions when creating large numbers of message monitor points in permanent monitor sets, as this will reduce the overall time required to create them. For more information on using transactions with LNS, see Chapter 4, *Programming an OpenLNS application*, of the *OpenLNS Programmer's Guide*. Monitor points in permanent monitor sets are not automatically removed when the application shuts down. If a monitor point is not going to be used again, you can remove it with the *Remove* method before closing your application. This does not apply to monitor points in temporary monitor sets, as temporary monitor sets are deleted as soon as the application that created them shuts down. As a result, you should use temporary monitor points for monitor points you will only need to use once. For more information on creating and using message monitor points, see the *Adding Message Monitor Points to a Monitor Set* section in Chapter 9 of the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects, and temporary monitor points,

	are not available on Independent clients.
Syntax	*msgMpObject* = *msgMpColl*.**Add** *msgMpName targetDevice connDesc* **Element** — **Description** *msgMpObject* — The newly defined *MsgMonitorPoint* object. *msgMpColl* — The *MsgMonitorPoints* collection object. *msgMpName* — The *Name* of the new *MsgMonitorPoint* object. *targetDevice* — The *AppDevice* or dynamic *MessageTag* object that the *MsgMonitorPoint* will monitor. *connDesc* — A *ConnectDescTemplate* object which determines how messages will be sent out through this monitor point. You can set this element to NULL to use the default LNS connection description template. **Note**: Message monitor points in temporary monitor sets do not support the use of connection description templates, and so you must set this element to NULL when adding a message monitor point to a temporary monitor set.
Added to API	LNS Release 3.0.

Remove

Summary	Removes a *MsgMonitorPoint* object from the system. This method can only be called when connected to the network database (after the *Network* object's *Open* method has been called). This method only applies to message monitor points in permanent monitor sets. When an OpenLNS application is closed, all temporary monitor sets and temporary message monitor points created by that application are deleted automatically.
Availability	Local, full, lightweight, and independent clients. Note that temporary monitor sets and temporary monitor points are not available on Independent clients.
Syntax	*msgMpColl*.**Remove** *indexName* **Element** — **Description** *msgMpColl* — The *MsgMonitorPoints* collection object containing the *MsgMonitorPoint* object to be removed.

	indexName	A Long value specifying the collection index of the *MsgMonitorPoint* object to remove, or a String value specifying the name of the *MsgMonitorPoint* object to remove.
Added to API	Prior to LNS Release 3.0.	

Properties

The *MsgMonitorPoints* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *MsgMonitorPoints* object in the *ConstClassIds* constant: **78 lcaClassIdMsgMonitorPoints** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.

Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *MsgMonitorPoint* object from a *MsgMonitorPoints* collection. You can retrieve a *MsgMonitorPoint* object from its *MsgMonitorPoints* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *MsgMonitorPoint* object in *MsgMonitorPoints* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** **Description** *retrievedObject* The *MsgMonitorPoint* object retrieved from the collection. *collObject* The *MsgMonitorPoints* collection object to be acted on. *index* A Long type specifying the ordinal index of the *MsgMonitorPoint* object to be retrieved. *stringExpression* A string type specifying the name of the *MsgMonitorPoint* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.

Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

Network

The Network object represents a single LONWORKS network. A network is a set of devices that interoperate via network variable and message tag connections. Network objects may be used to represent physically separate networks, or they may be used to represent multiple projects on the same physical network.

The following table summarizes the *Network* object.

Description	A single LONWORKS network.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Networks* collection object. *ObjectServer* object.
Default Property	*Name*.
Methods	• *Backup* • *CancelValidation* • *Close* • *CloseIndependent* • *CompactDb* • *CreateTemporaryMonitorSet* • *Open* • *OpenIndependent* • *PreReplace* • *Replace* • *Validate*
Properties	• *AllowPropagateModeDuringRemote* • *BitmapFilePath* • *Channels* • *ClassId* • *CurrentMonitorSets* • *DatabasePath* • *Description* • *EventInterval* • *Extensions* • *Handle* • *IconFilePath* • *IsOpen* • *IsOpenIndependent* • *MyVNI* • *Name* • *NetworkServiceDevices* • *NsiTimeout* • *OriginalName* • *Parent* • *RemoteNetworkName*

	• *ServerIdentifier* • *System*

Methods

The *Network* object contains the following methods.

- *Backup*
- *CancelValidation*
- *Close*
- *CloseIndependent*
- *CompactDb*
- *CreateTemporaryMonitorSet*
- *Open*
- *OpenIndependent*
- *PreReplace*
- *Replace*
- *Validate*

Backup

Summary	Make a backup copy of the network database, and exports it to a specified directory. The *System* object must be open when you invoke this method. If the *System* object is not open, the **LCA #67 lcaErrSystemNotOpen** exception will be thrown. You can backup the network database with this method at any time while the network is open and while clients are attached to it. However, if a remote Full client accesses the *Networks* collection while the database is being backed up, and that remote Fulll client does not already have open the network being backed-up, that network will not appear in the network collection. In addition, if a remote Full client attempts to open a network while it is being backed-up, the open may fail, whereas Local and Lightweight clients will simply wait for the backup to complete in this case. OpenLNS calls made by clients already connected to the database when a backup is initiated may not return until the backup is complete, and requests to modify the database will be suspended until the backup is complete. You should use this method to backup the network database before validating the database with the *Validate* method. You can then archive the backed-up database before performing any repairs that are discovered by the database validation procedure.
Availability	Local clients.
Syntax	*network.***Backup** *systemPath* **Element** **Description**

	network	The *Network* object to be acted upon.
	systemPath	The directory to which the backup is to be copied. The *systemPath* element must contain a valid system path. If the destination directory specified by this path does not exist, that directory will be created, as long as the first parent directory in the path exists. The directory permissions for this new folder will be the default permissions for the operating system on the computer you are using. If the directory specified by the *systemPath* element already exists, it must be empty. If there is any error during copying, or if an invalid system path is supplied, then the **LCA#159 lcaErrUnableToCreateBackup** exception is thrown. Before returning the error, all files copied by OpenLNS up to that point will be deleted from the destination directory, and the destination directory will be deleted if it was created by OpenLNS.
Added to API	LNS Release 3.20.	

CancelValidation

Summary	Cancels a database validation that is in progress. You can initiate a database validation by invoking the *Validate* method on the *Network* object. The *CancelValidation* method must be called from the same process that called the *Validate* method. Typically, it would be called from a *ProgressUpdate* method callback. If this method is called, and a database validation has not been previously initiated with the *Validate* method, it will fail silently. It may take several minutes to stop the validation process after you call the *CancelValidation* method. You can use the *ProgressUpdate* method to check the status of the process during this time period. When the validation has been stopped, the **LCA, #153 lcaErrDbValidationCancelled** exception will be thrown to notify the application.
Availability	Local clients.

Syntax	*network.***CancelValidation** **Element** **Description** *network* The *Network* object to be acted upon.
Added to API	LNS Release 3.20.

Close

Summary	Closes the OpenLNS network database and terminates monitor set monitoring. You should close the network's *System* object before invoking this method.
Availability	Local, full, and lightweight clients.
Syntax	*network.Close* **Element** **Description** *network* The *Network* object to be closed.
Added to API	Prior to LNS Release 3.0.

CloseIndependent

Summary	Closes the network database if the network database was opened using the *OpenIndependent* method. If this method is called when the network was opened using the *Open* method, an error will be returned.
Availability	Independent clients.
Syntax	*network.***CloseIndependent** **Element** **Description** *network* The *Network* object to be closed.
Added to API	Prior to LNS Release 3.0.

CompactDb

Summary	Defragments and re-indexes the OpenLNS network database. The method may not be called on a database that is open and in use by any client application. You should backup all databases before calling this method. Also, your computer should have at least twice as much free disc space as the size of the database when you call this method.
Availability	Local, full, lightweight, and independent clients.

Syntax	*network.***CompactDb** **Element** **Description** *network* The *Network* object to be acted on.
Added to API	Prior to LNS Release 3.0.

CreateTemporaryMonitorSet

Summary	Creates a temporary *MonitorSet* object. If you need monitor points that will only be used in a single client session, you should use temporary *MonitorSet* objects. Temporary monitor sets are opened automatically by OpenLNS as they are created, and can only be accessed from the client that created them. When a client releases a temporary monitor set, or when the client session in which a temporary monitor set was created ends, the temporary monitor set and all its monitor points are deleted. If you need to create a group of monitor points that you can use in multiple client sessions or that you intend to use multiple times, you should use the permanent *MonitorSet* objects described earlier in this section. However, if you do not need to re-use a monitor set, you should use temporary monitor sets because it takes less time and network resources to create them. This method is not available to Independent client applications. This method returns a *MonitorSet* object that is opened automatically upon creation, and closed as soon as the client session in which it was created ends. There are several other differences between temporary and permanent *MonitorSet* objects. For more information on these differences, see the *MonitorSet* object.
Availability	Local, full, and lightweight clients.
Syntax	*tempMonitorSet = network.CreateTemporaryMonitorSet* **Element** **Description** *tempMonitorSet* The newly defined temporary *MonitorSet* object. *network* The *Network* object to be acted upon.
Added to API	LNS Release 3.20.

Open

Summary	Opens the OpenLNS network database. Before this method can be used, the *Network* object must have been previously added using the *Add* method of the *ObjectServer*'s *Networks*

	collection. If this method is being invoked by a remote client, the application program must set the remote *NetworkInterface* for the *ObjectServer* (*ObjectServer*.**ActiveRemoteNI**) before invoking this method. Additionally, the OpenLNS Server must be running on the computer containing the OpenLNS Network Database. A network obtained from the *VNINetworks* collection cannot be opened with this method. It must instead be opened using the *OpenIndependent* method. Opening a network in *server-independent* allows the network to be monitored and controlled using *MonitorSet* objects without communicating with the OpenLNS Server. If you will be opening any networks with an OpenLNS application that is running as a Windows service, then the first application to *open* the OpenLNS Object Server must also be running as a Windows service. In addition, if a network is to be opened by an OpenLNS application that is running as Windows service, then that network and system must be opened by an OpenLNS application that is running as Windows service before it is opened with an OpenLNS application running as a user process. If you open the OpenLNS Object Server or a network with a user process before opening it with a service, you must first close the network and OpenLNS Object Server before opening them with an OpenLNS application that is running as Windows service. To avoid these problems, Echelon recommends that if an OpenLNS network is to be opened by a service, that service should be started automatically, and should open the network and systems that it will be accessing on startup, thus ensuring that the network is opened by the service before a user process.
Availability	Local, full, and lightweight clients.
Syntax	*network.Open* **Element** **Description** *network* The *Network* object to be opened.
Added to API	Prior to LNS Release 3.0.

OpenIndependent

Summary	Opens the OpenLNS network database in *server-independent* mode. You may only call this method from networks fetched from the *VNINetworks* collection. When a network is opened in server-independent mode, you can monitor and control monitor sets which have previously been defined on this

	computer without the OpenLNS Server running on the host. You may not access the network database. The same network should not be simultaneously opened in server-dependent and server-independent mode (with this method as well as the *Open* method). Doing this may cause unpredictable results. See the *ObjectServer* object's *Open* method for information on how to initialize the Object Server.
Availability	Independent clients.
Syntax	*network.OpenIndependent* **Element** **Description** *network* The *Network* object to be opened.
Added to API	LNS Release 3.0.

PreReplace

Summary	Allows a remote Full client to reattach to a network. In some situations, this may be necessary if the client's network interface was not made a permanent device on the Server computer. Under normal circumstances, when a remote full client re-opens a network, any network variables, connections and monitor sets created previously will still be available to the client, so long as the original NSD still exists in the database. To ensure that the NSD is never deleted from the OpenLNS database, the NSD must be configured as a permanent device on the network. You can do so by setting the *lcaNsdType* property of the *NetworkServiceDevice* object to **lcaNsdTypePermanent (1)**. However, under some circumstances, the correlation between the client and the NSD configuration may be lost, and this method may be used to re-associate the client with the correct NSD. For example, you will need to use this method if you open a network remotely from a new computer, and want that client to use the NSD information which was previously associated with another remote client computer (effectively moving the remote application and NSD configuration from one computer to another). An exception to this is if the original remote client used a layer 5 network interface, and you move the network interface to the new computer as well. In this case, OpenLNS will automatically associate the NSD in the database with the client based on the layer 5 network interface's neuron ID. You will also need to follow the procedure described below to reattach an NSD to a network if the network has been removed from the *RemoteNetworks* collection for the computer, and you are using a Layer 2 network interface (or

	if you install a new network interface on the computer). In all other scenarios (e.g. when operating as a Local client and upgrading the computer or network interface card, or when moving a layer 5 network interface card from one computer to another) you will not need to perform this procedure. To re-associate a remote Full client with the correct network service device and re-attach the client to the network, follow these steps: 1. Get the network service device to be attached to from the *NetworkServiceDevices* collection. 2. Call the *PreReplace* method with the selected network service device as an argument. 3. *Close* the network and release all references to the network. 4. Get the network and call the *Network* object's *Replace* method. 5. Call the *Network* object's *Open* method to open the network with all previously created monitor sets present.
Availability	Full clients.
Syntax	*networkObject*.**PreReplace** *sourceNSD* **Element** — **Description** *networkObject* — The *Network* object to be acted on. *sourceNSD* — The name of the *NetworkServiceDevice* object associated with the remote Full client that is to reattach to the network.
Added to API	LNS Release 3.0.

Replace

Summary	Completes the network service device replacement started by the *PreReplace* method. Under normal circumstances, when a remote full client re-opens a network, any network variables, connections and monitor sets created previously will still be available to the client, so long as original NSD still exists in the database. However, under some circumstances, the correlation between the client and the NSD configuration may be lost, and this method may be used in conjunction with the procedure described below to re-associate the client with the correct NSD. To ensure that the NSD is not deleted from the OpenLNS database, the NSD must be configured as a permanent device on the network. You can do so by setting the *lcaNsdType* property of the *NetworkServiceDevice* object to **lcaNsdTypePermanent (1)**. For example, you will need to use this method if you open a network remotely from a new computer, and want that client

	to use the NSD information which was previously associated with a remote client running on another computer (effectively moving the remote application and NSD configuration from one computer to another). An exception to this is if the original remote client used a layer 5 network interface, and you move the network interface to the new computer as well. In this case, OpenLNS will automatically associate the NSD in the database with the client based on the layer 5 network interface's neuron ID. You will also need to follow the procedure described below to reattach an NSD to a network if the network has been removed from the *RemoteNetworks* collection for the computer, and you are using a Layer 2 network interface (or if you install a new network interface on the computer). In all other scenarios (e.g. when operating as a Local client and upgrading the computer or network interface card, or when moving a layer 5 network interface card from one computer to another) you will not need to perform this procedure. To re-associate a client with the correct network service device and re-attach the client to the network, follow these steps: 1. Get the network service device to be attached to from the *NetworkServiceDevices* collection. 2. Call the *PreReplace* method with the selected network service device as an argument. 3. *Close* the network and release all references to the network. 4. Get the network and call the *Network* object's *Replace* method. 5. Call the *Network* object's *Open* method to open the network with all previously created monitor sets present.
Availability	Full clients.
Syntax	*networkObject*.**Replace** **Element** **Description** *networkObject* The *Network* object to be acted on.
Added to API	LNS Release 3.0.

Validate

Summary	Initiates a database validation on a network. When you invoke this method, OpenLNS will perform a consistency check on the network database, and report any inconsistencies or errors it finds. Inconsistencies that may be discovered during the database validation include orphan objects (objects that cannot be accessed through their parent object), broken interfaces, or duplicate objects. The *System* object must be open when you call this method. If

<table>
<tr><td></td><td>it is not open, the LCA, #67 lcaErrSystemNotOpen exception will be thrown.

You should consider using the Validate method to repair your network database a last resort. You should backup the database with the Backup method before performing a validation with repairs. For more information on this procedure, see the Backup method. It is safer for the client application to remove invalid objects from the database (delete and recreate a device) than for the database validation method to attempt to do so.

Depending on the size of the network database, it may take a considerable amount of time to complete the database validation. You can use the OnDbValidationEvent or the ProgressUpdate method to check the status of an ongoing database validation.

You can cancel a validation by invoking the CancelValidation method on the Network object. In this case, the method will not return a DatabaseValidationReport object.

The database validation must be initiated locally. While the database validation is in progress, clients will be unable to modify or write to the database. As a result, you should perform the database validation while a minimal number of client applications are connected to the database. One suggested approach is to backup the database and restore it with a different name and location, and perform the validation on this restored database. This approach will minimize the disruption caused by the validation, and has the benefit of producing in a backup whose validity is known.</td></tr>
<tr><td>Availability</td><td>Local clients.</td></tr>
<tr><td>Syntax</td><td>networkObject.Replace

Element — Description

validationReport — The DatabaseValidationReport object returned by the method. This object contains information describing the results of the database validation.

network — The Network object to be acted upon.

validationFlags — Specifies whether inconsistencies discovered during the database validation will be repaired.

The valid options for this element, which are contained in the ConstDbValidationFlags constant, are as follows:

0 lcaDbValidateOnly

OpenLNS performs the database</td></tr>
</table>

		validation without attempting to repair any of the errors it discovers. In this case, the *Validate* method returns a *DatabaseValidationReport* object summarizing the results of the database validation. **1 lcaDbValidateAndRepair** OpenLNS performs repairs on the errors and inconsistencies it discovers during the database validation. In this case, the *Validate* method returns a *DatabaseValidationReport* object summarizing the results of the database validation, and attempts to repair the problems it discovered during the validation. Not all error types can be repaired as part of the database validation process. You can use the *DatabaseValidationReport* object returned by the process to determine whether any errors remain in the database after the validation has completed.
	progressCallback	Optional callback interface to receive database validation progress events directly, rather than through the Object Server's *OnDbValidationEvent* event. This element will only take references to objects that are implemented as *ILcaProgressListener* or *_DLcaProgressListener* interface objects.
	validationReport	The *DatabaseValidationReport* object returned by the method. This object contains information describing the results of the database validation.
Added to API	LNS Release 3.20.	

Properties

The *Network* object contains the following properties:

- *AllowPropagateModeDuringRemote*
- *BitmapFilePath*

- *Channels*
- *ClassId*
- *CurrentMonitorSets*
- *DatabasePath*
- *Description*
- *EventInterval*
- *Extensions*
- *Handle*
- *IconFilePath*
- *IsOpen*
- *IsOpenIndependent*
- *MyVNI*
- *Name*
- *NetworkServiceDevices*
- *NsiTimeout*
- *OriginalName*
- *Parent*
- *RemoteNetworkName*
- *ServerIdentifier*
- *Systems*

AllowPropagateModeDuringRemote

Summary	Determines whether OpenLNS should change the *network management mode* to **lcaMgmtModePropagateConfigUpdates (0)** if it is necessary to open this network. This property applies to remote Full Client applications only. In some cases, OpenLNS cannot open remote networks when the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1).** This may be the case if the remote Full Client has not previously connected to the Object Server, if the remote Full Client has changed channels, or if changes have been made to the database such that commissioning the Network Service Device may cause inconsitencies in the configuration of physical devices on the network. This property must be set before you open the *System* object. If OpenLNS is unable to open a network due to the network management mode settings, the **NS, #31 lcaErrNsDeferConfigUpdatesMgmntMode** exception will be thrown. For more information on the network management mode, see the *MgmtMode* property.
Availability	Full clients.
Syntax	*network*.**AllowPropagateModeDuringRemoteOpen** = *flag* **Element** — **Description** *network* — The *Network* object being acted upon. *flag* — A Boolean value indicating whether OpenLNS should change the network management mode in

	order to successfully open this remote network. **TRUE**. OpenLNS changes the network management mode when necessary. Set this property to True if it is acceptable for OpenLNS to temporarily change the network management mode from **lcaMgmtModeDeferConfigUpdates (1)** to **lcaMgmtModePropagateConfigUpdates (0)** when opening a network from such a client application in these situations. When the network management mode is changed, all pending configuration updates will applied to the physical devices on the network. This may result in unwanted changes being propagated to the network. Once this has completed and the network has been opened, the management mode will be restored to **lcaMgmtModeDeferConfigUpdates (1)**. **FALSE**. OpenLNS does not change the network management mode.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.
Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** — **Description** *bmpFilePath* — The bitmap path and file name. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example,

	C:\MyBMPs\Object.BMP).
Added to API	Prior to LNS Release 3.0.

Channels

Summary	Contains the *Channels* collection object associated with the specified *Network* object.
Availability	Local, full, and lightweight clients.
Syntax	*channelsColl* = *networkObject*.**Channels** **Element** — **Description** *channelsColl* — The *Channels* collection object to be returned. *networkObject* — The *Network* object.
Data Type	*Channels* collection object.
Read/Write	Read only
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Network* object in the *ConstClassIds* constant: **1 lcaClassIdNetwork** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CurrentMonitorSets

Summary	Contains a collection of all of the *MonitorSet* objects that are

	commissioned into your client's NSD for a given network. The *CurrentMonitorSets* property returns a *MonitorSets* object that contains all the *MonitorSet* objects on that network that are currently stored in your client's NSD. This may be useful if you have created monitor sets while the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**. Although those monitor sets exist in the OpenLNS database and can also be accessed through the *MyVNI* property, they will not be commissioned into the NSD. As a result, they cannot be enabled or used for monitoring operations until the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)**. The collection accessed through the *CurrentMonitorSets* property only allows access to the monitor sets you can currently use on a network (the collection accessed through the *MyVNI* property allows access to these monitor sets, as well as those that have not yet been commissioned into your client's NSD). You can use all the monitor sets obtained through the *CurrentMonitorSets* property as runtime monitor sets, meaning that you can enable them and use them for monitoring operations. However, changes to their configuration are not allowed when accessed through this collection. You should the *MyVNI* property when you need to write to the configuration of your client's local *MonitorSet* objects.
Availability	Local, full, and lightweight clients.
Syntax	*monitorSets = network.CurrentMonitorSets* **Element** — **Description** *monitorSets* — The collection of *MonitorSet* objects on a network that are currently stored in your client's NSD. *network* — The *Network* object being acted upon.
Data Type	*MonitorSets* collection object.
Read/Write	Read only
Added to API	LNS Release 3.20.

DatabasePath

Summary	Contains the full path of the OpenLNS network database. Typically the full path for each OpenLNS network database is set when you add the network to the *Networks* collection. The database path is a parameter of the *Add* method, and can be a maximum of 230 characters long. See the *Add* method for the *Networks* object for more information.
Availability	Local, full, lightweight, and independent clients.

Syntax	*dbPath* = *object*.**DatabasePath** **Element** **Description** *dbPath* The full path of the global OpenLNS database. *object* The *Network* object.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *Network* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *Network* object. *object* The *Network* to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

EventInterval

Summary	The interval between event requests. This property sets the interval at which OpenLNS will call the *GetNextEvent* service for your Lightweight Client application. Each time LNS calls the *GetNextEvent* service, your Lightweight Client application will receive a TCP messag containing all the events it has registered for that have occurred since the last time the service was called. Consider a case where this property is set to the default value of 1 second. OpenLNS will call the *GetNextEvent* service every second. The application will receive the events it has registered for immediately after the service has been called, or as soon as the events occur after the service has been called. For example, say LNS calls the *GetNextEvent* service, but no events occur for another 750 ms. The application will receive those events at 750 ms, and will make the next GetNextEvent request 250 ms later. If no events occur before the interval defined by the

	EventInterval property expires, OpenLNS will not call the *GetNextEvent* service right away. Instead, it will wait until the next event (or set of events) occurs, deliver that event to the client, and then immediately call *GetNextEvent* (because the interval has already expired). Following that, OpenLNS will call the *GetNextEvent* service at the interval defined by this property. Note that as you set this property to lower and lower values, more and more request-response messages will be sent to the server. You should keep this in mind when setting this property, as it may cause more network traffic than you desire. This property is not applicable to Local Client and Full Client applications because the OpenLNS Server delivers events to those clients as soon as they occur. However, you can change the default value used by all of the Lightweight Clients on the network by changing the value of this property on a Local Client. By changing the value of this property on a Lightweight Client application, you change the value used by that client only.
Availability	Local, full, and lightweight clients.
Syntax	*eventInterval* = *networkObject*.**EventInterval** **Element** — **Description** *eventInterval* — The period of time that has passed since the last event request. *networkObject* — The *Network* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *Network* object. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer. **Note**: The *Extensions* collection for a *Network* object is stored in the specific OpenLNS global database computer, and it is not exported with the *Network* database. As a result, the collection would be lost in the process of transferring a network database from one computer to another.
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object.Extensions*

	Element	**Description**
	extensionsColl	The *Extensions* collection object.
	object	The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Handle

Summary	Contains the handle associated with the *Network* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property.	
Availability	Local, full, and lightweight clients.	
Syntax	*returnValue* = *object*.**Handle**	
	Element	**Description**
	returnValue	The NSS handle of the object.
	object	The object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

IconFilePath

Summary	Specifies the path and file name of an icon (*.**ICO** file) representation of the object.	
Availability	Local clients.	
Syntax	*IconFilePathFileName* = *object*.**IconFilePath**	
	Element	**Description**
	IconFilePathFileName	Icon file and path name
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32)	

	• Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

IsOpen

Summary	Indicates whether the specified *Network* object is currently open. You can open the *Network* object with the *Open* method, and you can close it with the *Close* method.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects are not available on Independent clients.
Syntax	*isOpenFlag* = *Object*.**IsOpe** **Element** — **Description** *isOpenFlag* — Boolean value. *Object* — A Boolean value indicating whether the *Network* object is currently open. **TRUE**. The *Network* object is currently open. **FALSE**. The *Network* object is currently closed.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

IsOpenIndependent

Summary	Indicates whether the specified *Network* object is currently open in *server-independent* mode. Successfully invoking the *OpenIndependent* method on a *Network* object will set its *IsOpen* property to True. Invoking the *CloseIndependent* method sets its *IsOpen* property to False.
Availability	Local, full, lightweight, and independent clients.
Syntax	*isOpenFlag* = *networkObject*.**IsOpenIndependent** **Element** — **Description** *isOpenFlag* — Boolean value. *Object* — A Boolean value indicating whether the *Network* object is currently open. **TRUE**. The *Network* object is currently open. **FALSE**. The *Network* object is currently closed.
Data Type	Boolean.

Read/Write	Read only.
Added to API	LNS Release 3.0.

MyVNI

Summary	Contains the *AppDevice* object which is used to create *MonitorSets* on this network. This *AppDevice* object is the only one that supports the *MonitorSets* property. You should access your monitor sets through this property when you plan to create new monitor sets, or when you plan to modify the configuration of an existing monitor set. For actual monitor and control operations, you should use the *CurrentMonitorSets* property.
Availability	Local, full, lightweight, and independent clients.
Syntax	*adObject= networkObject.MyVNI* **Element** — **Description** *adObject* — The *AppDevice* returned by the property. This *AppDevice* object represents the virtual network interface for this network. *network Object* — The *Network* object.
Data Type	*AppDevice* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object.***Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as

	they are added to the API.

NetworkServiceDevices

Summary	Contains the *NetworkServiceDevices* collection object associated with the specified *Network* object. This is the collection of all *NetworkServiceDevice* objects (the NSS and all NSIs) on the network.
Availability	Local, full, lightweight, and independent clients.
Syntax	*nsdColl = networkObject.NetworkServiceDevices* **Element** — **Description** *adObject* — The *AppDevice* returned by the property. This *AppDevice* object represents the virtual network interface for this network. *network Object* — The *Network* object.
Data Type	*NetworkServiceDevices* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NsiTimeout

Summary	Indicates how long the OpenLNS Object Server will wait for a client application to disconnect from the network before shutting down. Your application should close all networks before this interval expires when closing the Object Server.
Availability	Local, full, and lightweight clients.
Syntax	*timeoutValue = networkObject.***NsiTimeout** **Element** — **Description** *timeoutValue* — The NSI's timeout value in seconds. A value of **0** indicates that the system will not time out. The maximum value for this property is **32,767**. The default value for this property is **10** seconds. *networkObject* — The *Network* to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

OriginalName

Summary	Contains the network's original name as defined on the OpenLNS Server. The property may only be accessed after the remote client has connected to the OpenLNS Server using the system's *Open* method.
Availability	Local, full, and lightweight clients.
Syntax	*networkName* = *networkObject*.**OriginalName** **Element** — **Description** *networkName* — The returned network name. *networkObject* — The *Network* to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

RemoteNetworkName

Summary	Allows the name of the network as seen in the *RemoteNetworks* and *VNINetworks* to be changed. When a Full client network is opened for the first time on a computer, an entry is made in the *RemoteNetworks* and *VNINetworks* collections which identifies the network as `r_<Network Name>` (i.e. if the network name is `HVAC`, the name in the *RemoteNetworks* and *VNINetworks* collections will be `r_HVAC`). It is possible to change this name before opening the network for the first time(and thus creating the *RemoteNetworks* and *VNINetworks* entries) by setting this property immediately before the *Network* object's *Open* method is called. The new network name will be available in the *RemoteNetworks* and *VNINetworks* collections as soon as the *Open* method is called. If multiple Full client networks that use the same name will be opened from a computer, this allows them to be uniquely identified. Additionally, this allows the same network to have multiple *RemoteNetworks* and *VNINetworks* entries, each with a different name (and each containing its own collection of monitor sets).
Availability	Local, full, and lightweight clients.
Syntax	*networkObject*.**RemoteNetworkName** = *networkName* **Element** — **Description** *networkObject* — The *Network* object to be acted on. *networkName* — The new name of the network. This can be a maximum of 85 characters long. The network name may not include the following characxters: forward slash (/), back slash (\), period (.), and colon.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ServerIdentifier

Summary	Provides an identification string for the OpenLNS Server that has opened this network, which enables networks with the same name on different servers to be differentiated. When the *ObjectServer* object's *RemoteFlag* property is set to True before the *ObjectServer* is opened, the *Networks* collection will contain all of the remote full or lightweight client networks (see the *Flags* property) which have been opened by an OpenLNS Server application. It is possible that two or more of these networks opened with different OpenLNS Server applications will have the same name (multiple networks opened on a single OpenLNS Server must have unique names). This property allows these networks to be differentiated by provided a string representing a 4-byte hexadecimal value that uniquely identifies the LNS Sever.
Availability	Local, full, and lightweight clients.
Syntax	*serverId* = *networkObject*.**ServerIdentifier** **Element** — **Description** *serverId* — The unique 4-byte identifyer of the OpenLNS Server application which has opened this network. • If this *Network* object is accessed from the *RemoteNetworks* or *VNINetworks* collections, this property will contain "0000". • If this *Network* object is accessed from the *Networks* collection with the RemoteFlag property having been set to False, this property will contain "000". Once this value has been set, it is persistent through the OpenLNS Server and/or the application exiting and restarting. *networkObject* — The *Network* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Systems

Summary	Returns the *Systems* collection containing the *System* object. Only one *System* is allowed for each network (the *Systems* collection will always contain only one object).
Availability	Local, full, and lightweight clients.
Syntax	*systemsCollection* = *networkObject*.**Systems** **Element** — **Description** *systemsCollection* — Returned collection of *System* objects. *networkObject* — *Network* object to be acted upon.
Data Type	Systems collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Networks

The *Networks* object contains a collection of *Network* objects. This collection represents all of the networks known to the *ObjectServer*. The *ObjectServer* supports multiple active networks simultaneously, depending on the system resources available. The resources and memory required to support any number of open networks depends on the sizes of the networks, and the activities of the client application(s) operating on those networks. You should monitor the performance of your system when using large networks or multiple networks to ensure that you have the proper amount of memory available.

The following table summarizes the *Networks* object.

Description	A collection of *Network* objects.
Added to API	LNS Release 3.0.
Accessed Through	*ObjectServer* object.
Default Property	*Item*
Methods	• *Add* • *RemoveEx*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *Networks* object contains the following methods.

- *Add*
- *RemoveEx*

Add

Summary	Defines a new *Network* object. You can create a new network, or import an existing network.
Availability	Local, full, and lightweight clients.
Syntax	*networkObject* = *networksColl*.**Add**(*networkName, databasePath, createDatabase*)

Element	Description
networkObject	The newly defined *Network* object.
networksColl	The *Networks* collection object.
networkName	A String representing the name of the new network. This name is also applied to the network's *System* object, and can be no longer than 16 characters. If you enter a name that is longer than 16 characters, the **LCA, #23 lcaErrDatabasePathTooLong** exception will be thrown.
databasePath	A String containing the path for the network database. You can enter an absolute path or a relative path here. Full clients may not invoke this method. Lightweight clients may invoke the method, but no network is actually created. Instead, an entry is added within the client computer's server list. In this case, the *databasePath* parameter specifies the IP address and port for the server, e.g. "lns://myServer:2540", and the *createDatabase* parameter is ignored. Entries may be removed using the *Networks* collection object's *RemoveEx* method. You can specify the *dataBasePath* parameter as an absolute path or as a relative path. If you specify the *databasePath* parameter as a relative path, OpenLNS will automatically make it an absolute path by prepending the current working directory to it. For example, if you enter "Network01" as the *databasePath* element, and the current working directory is "C:\LM," the new database would be located in C:\LM\Network01" folder. **Note**: The complete database path

		can be no more than 230 characters long, including any characters OpenLNS adds to create an absolute path. Otherwise the **LCA, #23 lcaErrDatabasePathTooLong** exception will be thrown.
	createDatabase	A Boolean value indicating whether a new *Network* and a new OpenLNS database are to be created, or a new network is to be created based on an existing OpenLNS database. **TRUE**. Creates a new *Network* and a new OpenLNS network database. **FALSE**. Creates a new network based upon the existing database specified in the *databasePath* element. You can use this option to import a network from another computer, or restore a backed up network database.
Added to API	Prior to LNS Release 3.0.	

RemoveEx

Summary	Removes a *Network* object from the local *Networks* collection, and may delete the OpenLNS network database, removing all files that were associated with the network. This method includes an option for specifying whether the OpenLNS network database is to be deleted. If you do not delete the network database, you can restore the network later, without having to re-create the database. This may be useful if you want to store network databases on a central file server and add them to (or remove them from) any OpenLNS Server computer when needed.
Availability	Local clients.
Syntax	*networksObject*.**RemoveEx** *indexName, databaseOption* **Element** — **Description** *networksObject* — The *Networks* collection object to be acted upon. *indexName* — A Long value specifying the collection index of the *Network* object to remove, or a String value specifying the name of the *Network* object to remove. The *indexName* element used to identify the network to be deleted is a Variant type. This allows you to

		identify the network to be deleted by its name, which is stored in the *Name* property of the *Network* object, or by its index number within the *Networks* collection.
	databaseOption	Determines whether the network database is preserved or deleted. The possible values for this element, which are contained in the *ConstNetworkRemovalFlags* constant, are as follows: **1 lcaNetworkRemovalFlagLeaveFiles** Removes the entry for the selected network from the local *Networks* collection, but leave the network database files intact. This enables you to restore the network later, without having to re-create the network database. You can do so by calling the *Add* method on the network, with the *createDatabase* flag set to False to re-add the network, and the *databasePath* parameter set to point to the saved database. This may be useful in a system with a large number of networks, if you want to minimize the number of entries in the local *Networks* collection. **2 lcaNetworkRemovalFlagLeaveRegistry** Removes the entry for the selected network from the local *Networks* collection, but leaves the registry entries associated with the network database intact. **Note**: You can also specify **0** for this element to delete the network database, and all of its files and registry entries. In this case, the *RemoveEx* method works exactly as the deprecated *Remove* method.
Added to API	LNS Release 3.20.	

Properties

The *Networks* object contains the following properties:

- *ClassId*

- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Networks* object in the *ConstClassIds* constant: **2 lcaClassIdNetworks** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.

Item

Summary	Returns a *Network* object from a *Networks* collection. You can retrieve a *Network* object from its *Networks* collection by passing its index (ordinal position) within that *Networks* collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *Network* object in *Networks* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *Network* object retrieved from the collection. *collObject* — The *Networks* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *Network* object to be retrieved. *stringExpression* — A string type specifying the name of the *Network* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*.

	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

NetworkInterface

A *NetworkInterface* object represents a single instance of a LONWORKS Network Interface or Network Services Interface (for example, a U10/U20 USB network interface, or a PCLTA-21, PCLTA-20, or PCC-10 network interface card). The *Name* property contains the device name of the *NetworkInterface* (for example, LON1), which is set from the LonWorks Interfaces Control Panel application. The following table summarizes the *NetworkInterface* object.

Description	An instance of a LONWORKS network interface or network services interface.

Added to API	Prior to LNS Release 3.0.
Accessed Through	*NetworkInterfaces* collection object. *NetworkServiceDevice* object.
Default Property	*Name.*
Methods	None.
Properties	• *ClassId* • *Name* • *Parent*

Methods

The *NetworkInterface* object does not contain any methods.

Properties

The *NetworkInterface* object contains the following properties:

- *ClassId*
- *Name*
- *Parent*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *NetworkInterface* object in the *ConstClassIds* constant: **14 lcaClassIdNetworkInterface** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

NetworkInterfaces

A *NetworkInterfaces* object represents a collection of *NetworkInterface* objects. This collection contains all of the network interfaces on the local computer that are available to the Object Server. This collection is constructed from entries in the Windows registry.

Use the *Item* property to access a *NetworkInterface* object from this collection. For more network interface types, you can specify the network interface to be retrieved by its name, or by its index number within the *NetworkInterfaces* collection. However, remote network interfaces (RNIs) that connect to your application through the OpenLDV xDriver do not appear in the *NetworkInterfaces* collection until a session with that RNI has been fully established. This is not true if you are using the Default xDriver Profile, which uses the Windows Registry to ensure that all configured RNIs will be added to the *NetworkInterfaces* collection. For other Profiles, you can still create and access the appropriate *NetworkInterface* object by name from the collection using the *Item* property.

To do so, pass a string type specifying the name of the RNI to retrieve as the *stringExpression* element when you read the *Item* property. For xDriver network interfaces, the network interface name of the RNI device can be a maximum of 128 characters long, and must be specified using the following naming convention: x.*ProfileName*.*Downlink Lookup Key*.

- *ProfileName* represents the name of the xDriver Profile that will manage the connection to the RNI.
- *Downlink Lookup Key* represents the downlink lookup key assigned to the RNI in the xDriver database.

If you used the LonWorks Interfaces Control Panel application to configure the RNI, this is the name you assigned the RNI when you first created it with the application. For example, if the xDriver Profile name is myProfile and the downlink lookup key is RNI-0001, the network interface name would be “x.myProfile.RNI-0001”.

The following table summarizes the *NetworkInterfaces* object.

Description	A collection of *NetworkInterface* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*ObjectServer* object.
Default Property	*Item*.
Methods	None.
Properties	• *ClassId* • *Count* • *Item* • *_NewEnum*

Methods

The *NetworkInterfaces* object does not contain any methods.

Properties

The *NetworkInterfaces* object contains the following properties:

- *ClassId*

- *Count*
- *Item*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *NetworkInterfaces* object in the *ConstClassIds* constant: **15 lcaClassIdNetworkInterfaces** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *NetworkInterface* object from a *NetworkInterfaces* collection. You can retrieve a *NetworkInterface* object from its *NetworkInterfaces* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *NetworkInterface* object in *NetworkInterfaces* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *NetworkInterface* object retrieved from the collection. *collObject* — The *NetworkInterfaces* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *NetworkInterface* object to be retrieved. *stringExpression* — A string type specifying the name of the *NetworkInterface* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of

	foreach statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

NetworkResources

A *NetworkResources* contains information related to the network resources allocated on the system. The properties of the *NetworkResources* object indicate how many exclusive and sharable pool selectors are available on the system, how many group IDs and subnets have been allocated on the system, and how many *AppDevices* and *Routers* have been installed on the system.

This information may be especially useful if you are managing a large system. For example, if you are writing an application that creates large numbers of multicast connections on a system, you will need to know how many exclusive selectors are available on the system. Another example would be if you are merging two OpenLNS databases, you will need to know how many subnets and exclusive selectors have been assigned in each database, to make sure that the merged database will not exceed the limits for each property.

Description	Contains information related to the network resources allocated on the system.
Added to API	LNS Release 3.20.
Accessed Through	*System* object.
Default Property	None.
Methods	None.
Properties	• *AppDeviceCount* • *ClassId* • *ExclusiveSelectorPoolSize* • *ExclusiveSelectorsAvailable* • *GroupIdsAllocated* • *Parent* • *RouterCount* • *SharableSelectorPoolSize* • *SubnetsAllocated*

Methods

The *NetworkResources* object does not contain any methods.

Properties

The *NetworkResources* object contains the following properties:

- *AppDeviceCount*
- *ClassId*
- *ExclusiveSelectorPoolSize*
- *ExclusiveSelectorsAvailable*
- *GroupIdsAllocated*
- *Parent*
- *RouterCount*
- *SharableSelectorPoolSize*
- *SubnetsAllocated*

AppDeviceCount

Summary	Indicates the number of *AppDevice* objects that have been installed on the system. This includes all devices that are installed in the OpenLNS database, and managed by OpenLNS. This does not include *AppDevices* contained in the *Discovered.Uninstalled subsystem*. This count does not include routers. You can determine how many routers have been installed on the system by reading the *RouterCount* property.
Availability	Local, full, and lightweight clients.
Syntax	*count* = *networkResources*.**AppDeviceCount** **Element** — **Description** *count* — The number of *AppDevices* installed on the *System*. The valid range for this property is 1–32,385. *networkResources* — The *NetworkResources* object being acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description**

	classIdValue	The object class of the object. The following value is defined for the *NetworkResources* object in the *ConstClassIds* constant: **93 lcaClassIdNetworkResources**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

ExclusiveSelectorPoolSize

Summary	Indicates the total number of exclusive selectors in the system, including those that are already being used by connections. Typically, this value will be set to 11,264. A network variable selector is a 14-bit number used to identify connected network variables. Each connection in the system is assigned a selector value, and all network variables in a given connection must use the same selector. Intersecting connections must also use the same network variable selector. OpenLNS can also share network variable selectors among unicast connections if those connections are between disjoint devices. Network variable selectors that are shared between unrelated connections come from the pool of sharable selectors available on the system. OpenLNS assigns exclusive network variable selectors to sets of intersecting connections, and to each connection that contains multiple targets. The selectors used to form this type of connection are called exclusive selectors. You can determine exactly how many exclusive selectors have not been assigned, and are still available, by reading the *ExclusiveSelectorsAvailable* property.	
Availability	Local, full, and lightweight clients.	
Syntax	*count* = *networkResources*.**ExclusiveSelectorPoolSize**	
	Element	**Description**
	count	The total number of exclusive selectors in the system, including those that are already being used by connections.
	networkResources	The *NetworkResources* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	

Added to API	LNS Release 3.20.

ExclusiveSelectorsAvailable

Summary	Indicates the total number of exclusive selectors that are currently available in the system. A network variable selector is a 14-bit number used to identify connected network variables. Each connection in the system is assigned a selector value, and all network variables in a given connection must use the same selector. Intersecting connections must also use the same network variable selector. OpenLNS can also share network variable selectors among unicast connections if those connections are between disjoint devices. Network variable selectors that are shared between unrelated connections come from the pool of sharable selectors available on the system. LNS assigns exclusive network variable selectors to sets of intersecting connections, and to each connection that contains multiple targets. The selectors used to form this type of connection are called exclusive selectors. You can find out how many exclusive selectors exist on the system (including those that are already in use) by reading the *ExclusiveSelectorPoolSize* property.
Availability	Local, full, and lightweight clients.
Syntax	*count* = *networkResources*.**ExclusiveSelectorsAvailable** **Element** — **Description** *count* — The number of exclusive selectors available in the system. *networkResources* — The *NetworkResources* object being acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

GroupIdsAllocated

Summary	Indicates the number of LonTalk group IDs that have been allocated on the system. Each set of connections in the system that use multicast addressing must have a unique group ID. As a result, the number of group IDs that have been allocated is critical to networks with large numbers of connections and devices. This property could be used by a network management application to determine if the multicast connections that it's creating are efficiently using available network resources. You can also use this property when merging two OpenLNS databases to determine if they have more than a combined total of 256 group IDs allocated (in which case, the merge would fail). You can limit the use of multicast addressing with your applications using the *AliasOptions* and *BroadcastOptions* properties of the *ConnectDescTemplate* object used by the connection.
Availability	Local, full, and lightweight clients.
Syntax	*count* = *networkResources*. **GroupIdsAllocated** **Element** — **Description** *count* — The number of group IDs allocated on the system. This property has a range of 0–256, as each system supports a total of 256 group IDs for multicast addressing. *networkResources* — The *NetworkResources* object being acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*.

	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

RouterCount

Summary	Indicates the number of *Router* objects that have been installed on the system. This includes all routers that are installed in the OpenLNS database, and managed by OpenLNS. This does not include *Routers* contained in the *Discovered.Uninstalled subsystem*.	
Availability	Local, full, and lightweight clients.	
Syntax	*count* = *networkResources.RouterCount*	
	Element	**Description**
	count	The number of routers installed on the *System*. The valid range for this property is 1–16,192.
	networkResources	The *NetworkResources* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

SharableSelectorPoolSize

Summary	Indicates the total number of selectors in the system that can be used by more than one non-intersecting connection. Typically, this property will be set to 1,024. A network variable selector is a 14-bit number used to identify connected network variables. Each connection in the system is assigned a selector value, and all network variables in a given connection must use the same selector. Intersecting connections must also use the same network variable selector. OpenLNS can also share network variable selectors among unicast connections if those connections are between disjoint devices. Network variable selectors that are shared between unrelated connections come from the pool of sharable selectors available on the system.	
Availability	Local, full, and lightweight clients.	
Syntax	*count* = *networkResources*.**SharableSelectorPoolSize**	
	Element	**Description**

	count	The number of selectors that can be used by more than one connection in the system.
	networkResources	The *NetworkResources* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

SubnetsAllocated

Summary	Indicates the number of subnets that have been allocated on the system. Subnets are the second component of the three-component LonTalk domain/subnet/device addressing hierarchy used by devices on LONWORKS networks. The subnet address is the level at which routers decide whether to forward a packet; therefore, the same subnet cannot appear on both sides of a configured or learning router. Subnets are typically added to the system automatically as routers or devices are added. This property has a range of 1–255, as there can be a maximum of 255 subnets per system. Knowing how many subnets have been allocated may be useful if you are managing large systems. For example, if you plan to merge two OpenLNS databases, you can use this property to determine if they have more than a combined total of 255 subnets (in which case, the merge would fail).	
Availability	Local, full, and lightweight clients.	
Syntax	*count* = *networkResources*.**SubnetsAllocated**	
	Element	**Description**
	count	The number of subnets that have been allocated on the system.
	networkResources	The *NetworkResources* object being acted upon.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.20.	

NetworkServiceDevice

A *NetworkServiceDevice* (NSD) object represents a single instance of an OpenLNS Server or a Network Services Interface (NSI). The following table summarizes the *NetworkServiceDevice* object.

Description	A single OpenLNS Server or NSI.

Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object. *NetworkServiceDevices* object. *System* object.
Default Property	*Name*.
Methods	• *BeginResetEvent* • *EndResetEvent*
Properties	• *AppDevice* • *BitmapFilePath* • *ClassId* • *DefaultApplication* • *Description* • *Dialup* • *Extensions* • *IconFilePath* • *Interfaces* • *LcaNsdType* • *MipIsLayer2* • *Name* • *NetworkInterface* • *NetworkInterfaceFlag* • *NodeHandle* • *NsiHandle* • *NsiNodeId* • *NsiSubnetId* • *NssFlag* • *Parent* • *PingClass*

Methods

The *NetworkServiceDevice* object contains the following methods:

- *BeginResetEvent*
- *EndResetEvent*

BeginResetEvent

Summary	Enables the *OnNetworkServiceDeviceResetNew* event for a network service device.
Availability	Local, full, and lightweight clients.
Syntax	*nsdObject.BeginResetEvent* **Element** — **Description** *nsdObject* — The *NetworkServiceDevice* object.
Added to API	Prior to LNS Release 3.0.

EndResetEvent

Summary	Disables the *OnNetworkServiceDeviceResetNew* event.
Availability	Local, full, and lightweight clients.
Syntax	*nsdObject.EndResetEvent* **Element** **Description** *nsdObject* The *NetworkServiceDevice* object.
Added to API	Prior to LNS Release 3.0.

Properties

The *NetworkServiceDevice* object contains the following methods:

- *AppDevice*
- *BitmapFilePath*
- *ClassId*
- *DefaultApplication*
- *Description*
- *Dialup*
- *Extensions*
- *IconFilePath*
- *Interfaces*
- *LcaNsdType*
- *MipIsLayer2*
- *Name*
- *NetworkInterface*
- *NetworkInterfaceFlag*
- *NodeHandle*
- *NsiHandle*
- *NsiNodeId*
- *NsiSubnetId*
- *NssFlag*
- *Parent*
- *PingClass*

AppDevice

Summary	Identifies a host application using the *NetworkServiceDevice* as a network interface. An *AppDevice* object is automatically created and assigned to this property when the *NetworkServiceDevice* object is created. One *NetworkServiceDevice* is created when the OpenLNS database is created, and additional Network Service Devices are created to support remote full clients (usually one per remote computer that accesses the OpenLNS database).
Availability	Local, full, and lightweight clients.
Syntax	*appDevObject = nsdObject.AppDevice*

	Element	**Description**
	appDevObject	The *AppDevice* object to be returned.
	nsdObject	The *NetworkServiceDevice* object to be acted on.
Data Type	*AppDevice* object.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.	
Availability	Local clients.	
Syntax	*bmpFilePath* = *object*.**BitmapFilePath**	
	Element	**Description**
	bmpFilePath	The bitmap path and file name.
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).	
Added to API	Prior to LNS Release 3.0.	

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).	
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.	
Syntax	*classIdValue* = *object*.**ClassId**	
	Element	**Description**
	classIdValue	The object class of the object. The following value is defined for the *NetworkServiceDevice* object in the

	ConstClassIds constant: **40 lcaClassIdNetworkServiceDevice** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

DefaultApplication

Summary	Contains the default application information for this *NetworkServiceDevice* object. The *Application* object gives the application's name, state, version number, and the version of the API that compiled the application.
Availability	Local, full, and lightweight clients.
Syntax	*defaultApp* = *nsdObject*.**DefaultApplication** **Element** **Description** *defaultApp* The *Application* object to be returned. *nsdObject* The *NetworkServiceDevice* object to be acted on.
Data Type	Application.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Description

Summary	Stores description information about the *NetworkServiceDevice* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *NetworkServiceDevice* object. *object* The *NetworkServiceDevice* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as

	soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Dialup

Summary	Specifies whether the *NetworkServiceDevice* is communicating through a modem. This property allows applications to be designed such that they can check to see if communication with a device is taking place over a modem before making permanent connections or other communications decisions.
Availability	Local, full, and lightweight clients.
Syntax	d*ialupValue* = *nsdObject*.**Dialup** **Element** — **Description** *dialupValue* — A Boolean value. **TRUE**. The *NetworkServiceDevice* is communicating through a dialup modem. **FALSE**. The *NetworkServiceDevice* is not communicating through a dialup modem. *nsdObject* — The *NetworkServiceDevice* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *NetworkServiceDevice* object. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer.
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object.Extensions* **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (***.ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** — **Description** *IconFilePathFileName* — Icon file and path name *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

Interfaces

Summary	Contains the *Interfaces* collection object associated with the specified *NetworkServiceDevice* object. This is the collection of virtual and custom interfaces associated with the device. The *Interfaces* collection allows virtual and custom interfaces to be added to any device that supports dynamic network variables, dynamic message tags, or dynamic LonMarkObjects. You add custom interfaces to a device with the *Add* method.
Availability	Local, full, and lightweight clients.
Syntax	*interfaceColl* = *object.Interfaces* **Element** — **Description** *interfaceColl* — The *Interfaces* collection to be returned. *object* — The *NetworkServiceDevice* object to be acted on.
Data Type	*Interfaces* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LcaNsdType

Summary	Specifies the type of network service device, which determines whether its NSI is removed when the NSI client closes.
Availability	Local, full, and lightweight clients.
Syntax	*interfaceColl = object.Interfaces* **Element** — **Description** *nsdType* — A Long value indicating the NSD type. The valid values for this constant, which are contained in the *ConstLcaNsdType* constant, are as follows: **0 lcaNsdTypeStandard** The NSI is removed unless it is a dial-up interface or it is involved in connections. **1 lcaNsdTypePermanent** The NSI is not removed. **2 lcaNsdTypeTransient** The NSI is always removed. Any existing connections will be disconnected. *nsdObject* — The *NetworkServiceDevice* to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

MipIsLayer2

Summary	Indicates whether this network service device object represents a device with an OpenLNS high-performance network interface.
Availability	Local, full, and lightweight clients.
Syntax	*isLayer2Flag = nsdObject*.**MipIsLayer2** **Element** — **Description** *isLayer2Flag* — A Boolean value. **TRUE**. The *NetworkServiceDevice* is using an OpenLNS high-performace network interface. **FALSE**. The *NetworkServiceDevice* is

	not using an OpenLNS high-performace network interface. *nsdObject* — The *NetworkServiceDevice* object from which to get the information.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.),colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NetworkInterface

Summary	Contains the *NetworkInterface* object associated with the specified *NetworkServiceDevice* object.
Availability	Local, full, and lightweight clients.
Syntax	*niObject* = *nsdObject*.*NetworkInterface* **Element** — **Description** *niObject* — The *NetworkInterface* object associated with the network service device. *nsdObject* — The *NetworkServiceDevice* object to be acted upon.
Data Type	*NetworkInterface* object.

Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

NetworkInterfaceFlag

Summary	Specifies whether the *NetworkInterface* has been set.
Availability	Local, full, and lightweight clients.
Syntax	*netInterfaceFlag* = *nsdObject*.**NetworkInterfaceFlag** **Element** — **Description** *netInterfaceFlag* — A Boolean value. **TRUE**. The *NetworkInterface* property in the specified *NetworkServiceDevice* has been set to a valid interface object. **FALSE**. The *NetworkInterface* property is not valid and should not be used. *nsdObject* — The *NetworkServiceDevice* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NodeHandle

Summary	The unique ID assigned to the *NetworkServiceDevice* by the NSS. This property is equivalent to the *Handle* property of an *AppDevice* object. If this *NetworkServiceDevice* has an *AppDevice* associated with it, the *Handle* of that *AppDevice* will be equal to the *NodeHandle* of the *NetworkServiceDevice*. An OpenLNS application would only use the *NodeHandle* to directly invoke services of the LNS Host API.
Availability	Local, full, and lightweight clients.
Syntax	*handleValue* = *nsdObject*.**NodeHandle** **Element** — **Description** *handleValue* — The handle of the node as a long. *nsdObject* — The *NetworkServiceDevice* object to be acted on.
Data Type	Long.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

NsiHandle

Summary	Returns the unique identifier assigned to an NSI by the OpenLNS Server. This handle is different than the *NodeHandle* and *Handle* properties.
Availability	Local, full, and lightweight clients.
Syntax	*handleValue* = *nsdObject*.**NsiHandle** **Element** — **Description** *handleValue* — The NSI's handle. *nsdObject* — The *NetworkServiceDevice* to be acted on
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NsiNodeId

Summary	Contains the node ID associated with the network services interface (NSI). The *NsiNodeId* and *NsiSubnetId* omprise the logical network subnet/node address the NSD will use to send subnet node and group addressed messages, if the NSD is using a high-performance network interface. This address is assigned to the network services interface when the *Open* method is invoked on the *System* object. The *System* object must be open for these properties to contain a valid value. The *SubnetId* and *NodeId* properties of the *AppDevice* object contained by the *NetworkServiceDevice* return the subnet/node address the NSD will use for broadcast and neuron ID addressing. Standard network interfaces use the *NsiSubentId* and *NsiNodeId* only for server/remote full client communication.
Availability	Local, full, and lightweight clients.
Syntax	*idValue* = *nsdObject*.**NsiNodeId** **Element** — **Description** *idValue* — The NSI's node ID. *nsdObject* — The *NetworkServiceDevice* to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NsiSubnetId

Summary	Contains the subnet ID associated with the network services interface (NSI). The *NsiNodeId* and *NsiSubnetId* comprise the logical network subnet/node address the NSD will use to send subnet node and group addressed messages, if the NSD is using a high-performance network interface. This address is assigned to the network services interface when the *Open* ethod is invoked on the *System* object. The *System* object must be open for these properties to contain a valid value. The *SubnetId* and *NodeId* properties of the *AppDevice* object contained by the *NetworkServiceDevice* return the subnet/node address the NSD will use for broadcast and neuron ID addressing. Standard network interfaces use the *NsiSubentId* and *NsiNodeId* only for server/remote full client communication.
Availability	Local, full, and lightweight clients.
Syntax	*idValue* = *nsdObject*.**NsiSubnetId** **Element** — **Description** *idValue* — The NSI's node ID. *nsdObject* — The *NetworkServiceDevice* to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NssFlag

Summary	Indicates whether this *NetworkServiceDevice* is the NSD for the NSS engine.
Availability	Local, full, and lightweight clients.
Syntax	*nssFlag* = *nsdObject*.**NssFlag** **Element** — **Description** *nsdObject* — The *NetworkServiceDevice* object to be acted on. *nssFlag* — A Boolean value. **TRUE**. The NSD is used by the NSS engine. **FALSE**. This NSD is used by an OpenLNS Remote Client.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PingClass

Summary	Determines the frequency with which a device is tested (pinged) to determine if it is still attached to the network. This property classifies devices based on the probability that the device may be detached. The higher the probability, the more frequently the device will be pinged. The Object Server assumes a device to be detached if it cannot communicate with that device three consecutive times.
Availability	Local, full, and lightweight clients.
Syntax	*pingClassValue* = *Object*.**PingClass** **Element** — **Description** *Object* — The device object to be acted on. *pingClassValue* — The ping class of this object. The valid values for this element, which are provided in the *ConstPingClass* constant, are as follows: **0 lcaPingClassDefault** If this value is written to the PingClass property, OpenLNS will use the default **lcaPingClassStationary (3)** value. **1 lcaPingClassMobile**

		Class for nodes which move frequently. **2 lcaPingClassTemporary** Class for temporary nodes. **3 lcaPingClassStationary** Class for nodes which rarely move. This is the default value. **4 lcaPingClassPermanent** Class for nodes which never move. **Note**: You change the ping interval that applies to each class with the *System* object's *PingIntervals* property.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

NetworkServiceDevices

The *NetworkServiceDevices* object represents a collection of *NetworkServiceDevice* objects. The instance of this collection accessed through a *Network* object contains all of the network service devices attached to the network both locally (in this computer) and remotely. These objects are managed by the Object Server; therefore, you do not have to explicitly add or delete items from this collection.

The following table summarizes the *NetworkServiceDevices* object.

Description	A collection of *NetworkServiceDevice* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Network* object.
Default Property	*Item*.
Methods	• *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *NetworkServiceDevices* object contains the following method.

- *Remove*

Remove

Summary	Removes an object from the specified collection.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** — **Description** *objectColl* — The collection containing the object to be removed. *name* — A Long value specifying the collection index of the object to remove, or a String value specifying the name of the object to remove.
Added to API	Prior to LNS Release 3.0.

Properties

The *NetworkServiceDevices* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *NetworkServiceDevices* object in the *ConstClassIds* constant: **41 lcaClassIdNetworkServiceDevices** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *NetworkServiceDevice* object from a *NetworkServiceDevices* collection. You can retrieve a *NetworkServiceDevice* object from its *NetworkServiceDevices* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *NetworkServiceDevice* object in *NetworkServiceDevices* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *NetworkServiceDevice* object retrieved from the collection. *collObject* — The *NetworkServiceDevices* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *NetworkServiceDevice* object to be retrieved.

	stringExpression A string type specifying the name of the *NetworkServiceDevice* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* **=** *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.

Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

NetworkVariable

A *NetworkVariable* object represents a single instance of a network variable.

This object is used to represent a network variable that actually appears in an *AppDevice* object's interface, or that is part of a *DeviceTemplate* program interface definition.

- If a *NetworkVariable* object is accessed through a *DeviceTemplate* object, it represents a network variable that is part of a program interface definition. In this case, some of the methods and properties do not apply to the object. For example, it is not possible to read the value of a network variable that is accessed through a *DeviceTemplate* .
- If a *NetworkVariable* object is accessed through the *Interface* property of an *AppDevice* object, it represents a network variable defined on that device's interface. It may also represent a connection hub, depending on how you have used the network variable in connections. The *NetworkVariable* objects that are being used as connection hubs can be accessed through the *NetworkVariables* collection contained by the *NVHubs* property of an *AppDevice* object. You can use this collection to determine which network variables on a device are being used as connection hub.

The following table summarizes the *NetworkVariable* object.

Description	A single network variable.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*NetworkVariables* collection object.
Default Property	*Name*
Methods	• *AddTarget* • *Connect* • *Disconnect* • *DsRestoreOptions* • *DsSaveOption* • *GetDataPoint* • *MoveToInterface* • *ToString*
Properties	• *Aliases* • *AppDevice* • *AppDeviceName* • *AuthenticationConfigFlag*

	• *AuthenticationFlag* • *ChangeableTypeSupport* • *ClassId* • *ConfigClassFlag* • *ConfigProperties* • *ConnectDescTemplate* • *Description* • *Direction* • *DsFormatType* • *DsIsDefaultFormat* • *DsPollInterval* • *DsPriority* • *EstimatedMaxRate* • *EstimatedRate* • *Extensions* • *FuncProfileDescription* • *FuncProfileName* • *FuncProfileProgrammaticName* • *ImplementsCp* • *Index* • *IsConfigProperty* • *IsDynamic* • *IsPolled* • *Length* • *LmNumberManufacturerAssigned* • *LonMarkMemberIndex* • *LonMarkMemberNumber* • *LonMarkObjectNumber* • *MaxLength* • *Name* • *NVHubs* • *NVTargets* • *OfflineFlag* • *Parent* • *ParentInterface* • *Priority* • *PriorityConfigFlag* • *ProgrammaticName* • *Selector* • *SelfDocumentation* • *ServiceType* • *ServiceTypeConfigFlag* • *SnvtId* • *SnvtTypeIsModifiable* • *SyncFlag* • *TypeSpec*

Methods

The *NetworkVariable* object contains the following methods.

- *AddTarget*
- *Connect*

- *Disconnect*
- *DsRestoreOptions*
- *DsSaveOption*
- *GetDataPoint*
- *MoveToInterface*
- *ToString*

AddTarget

Summary	Adds a single *NetworkVariable* to a hub's pending target list. This method defines a hub network variable's pending target list. This list is used when the *Connect* or *Disconnect* method is invoked to create or remove a network variable connection. *Connections*, as defined within OpenLNS, always consist of a single hub and one or more complementary targets. To create a network variable connection, follow these steps: 1. Select a single hub object and a set of one or more target objects to connect to the hub. 2. Add the targets to the pending target list by invoking the hub's *AddTarget* method for each target object (up to a maximum of 25, see below). 3. When the list is complete, invoke the hub's *Connect* method. To remove a connection, invoke the *Disconnect* method, instead. The pending target list will only hold 25 targets at a time, but it is cleared upon completion of the *Connect* or *Disconnect* method. You can therefore create larger connections by iterating through the process outlined above. For example, upon completion of the *Connect* method, you can add additional targets by invoking the *AddTarget* method on the original hub object. You can then invoke the *Connect* method to append the new targets to the previously defined connection. For network variable connections, the hub and target variables must be complementary. A complementary network variable is one that has a matching type/length but the opposite direction. For example, if the hub is an output network variable, all the targets must be input network variables. The type or length restriction is applied depending on the category of network variable used. When connecting standard network variable types (SNVTs), all members of must be the same type. When connecting user-defined network variable types (UNVTs), all members must be the same length. While its basic connection model is fairly simple, LNS allows network variables to participate in multiple connections. As a result, it is possible to create arbitrarily complex network

	variable connections on a LonWorks network (subject to the constraints of the LonTalk protocol). This is accomplished by calling the *AddTarget* and *Connect* method with multiple hubs and overlapping targets. A consequence of the superposition of connections is that a network variable may find itself in a "mirrored" connection. This situation occurs when a network variable A is the hub of a connection containing target network variable B., **and** B is the hub of a connection containing A. (Connection segment AB is mirrored by BA). When removing connections, you must consider that the network variables will remain bound until both connections are removed.
Availability	Local, full, and lightweight clients.
Syntax	*nvMtObject.AddTarget targetObject* **Element** — **Description** *nvMtObject* — The hub *NetworkVariable* object. *targetObject* — The *NetworkVariable* object to be added to the target list.
Added to API	Prior to LNS Release 3.0.

Connect

Summary	Connects a hub network variable to the network variables contained in the hub's pending target list. This method creates a new connection or adds to an existing one. The connection consists of the hub and its targets. The hub is the network variable object upon which the method is invoked. Before invoking the *Connect* method, one or more targets must be added to the hub's pending target list using the *AddTarget* method. When the method is invoked, the OpenLNS Server defines the connection (using the parameters specified in the hub's *ConnectDescTemplate* object) and, if the *MgmtMode* property is set to **lcaMgmtModePropagateConfigUpdates (0)**, it connects the objects on the network, then clears the hub's pending target list. As part of the connection process, the OpenLNS Server updates the hub object's *NVTargets* property, as appropriate. If a new connection is created, the hub object is also added to the appropriate *NetworkVariables* property of the system's *Connections* object. When you create large or complex connections that require calling the *Connect* method more than once, you should use the *StartTransaction* and *StartTransaction* methods to group the calls into a single transaction. You can use the *OnNodeConnChangeEvent* to track when

	connections are created or modified with this method.
Availability	Local, full, and lightweight clients.
Syntax	*nvObject.Connect* **Element** **Description** *nvObject* The hub *NetworkVariable* object to be connected.
Added to API	Prior to LNS Release 3.0.

Disconnect

Summary	Removes a hub or network variable's entire connection or disconnects the network variables contained in the hub's pending target list. The behavior of this method is dependent upon the state of the hub's pending target list, which was created by the *AddTarget* method. • If the pending target list is empty, the method disconnects all members of the connection (all network variables contained within the hub's *NVTargets* property). • If elements have been added to the pending target list, the method disconnects those particular elements and clears the pending target list. When all targets have been disconnected from a hub network variable, the connection ceases to exist. The hub is subsequently deleted from the appropriate *NetworkVariables* property of the system's *Connections* object. You can use the *OnNodeConnChangeEvent* to track when connections are removed with this method.
Availability	Local, full, and lightweight clients.
Syntax	*object.Disconnect* **Element** **Description** *object* The *NetworkVariable* object to be acted on.
Added to API	Prior to LNS Release 3.0.

DsRestoreOptions

Summary	Restores the data server options saved in the persistent database using the *DsSaveOptions* method, overwriting any values which had been set during the current session. Options that were modified by this method will not take affect until the data server is paused and restarted, although they can be read by fetching the appropriate property. This method will restore all *NetworkVariable* data server

	options.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**DsRestoreOptions** **Element** **Description** *object* The *NetworkVariable* object to be acted on.
Added to API	Prior to LNS Release 3.0.

DsSaveOptions

Summary	Saves the current set of data server options to the persistent database. Data server options are not persistent for *NetworkVariable* objects. For new option values to persist across data object instantiations, you must invoke the *DsSaveOptions* method. This method saves the values which were most recently set, not the values currently in use. This distinction arises because the data server must be paused and restarted before modified options can take affect. This method will save all *NetworkVariable* data server options.
Availability	Local, full, and lightweight clients.
Syntax	*object.DsSaveOptions* **Element** **Description** *object* The *NetworkVariable* object to be acted on.
Added to API	Prior to LNS Release 3.0.

GetDataPoint

Summary	Returns a *DataPoint* object that you can use to read or write to the value of the network variable. You should use data points to read and write to the values of your network variables. The reason for this is that when you create a data point with the *GetDataPoint* method, your application will have sole access to that data point, and it will manage the format of the data contained in the network variable locally. As a result, it will avoid misinterpreting any formatting changes that may be made to a network variable's value by other client applications. **Note**: The *Value* property, which was used previously to read and write network variables, was deprecated in LNS Release 3.20.
Availability	Local, full, and lightweight clients.

Syntax	*dpObject=networkVariable.DataPoint options* **Element** — **Description** *dpObject* — The *DataPoint* object returned. *networkVariable* — The *NetworkVariable* object being acted upon. *options* — This field is fixed at **0**, and it is reserved for future use.
Added to API	LNS Release 3.20.

MoveToInterface

Summary	Moves a dynamic network variable from one custom interface on a device to another. Each *AppDevice* on a network includes an *Interface* property that contains the device's main interface, and an *Interfaces* collection that contains the custom interfaces that have been added to the device dynamically. The interfaces each contain LonMarkObjects and network variables that reflect the device's functionality on the network. You can use the *MoveToInterface* method to move a dynamic network variable or *LonMarkObject* from one custom interface on a device to another. An advantage of this is that you do not have to delete the network variable or *LonMarkObject* from the first custom interface, and then add it back to the second one. Another advantage of this method is that you can use it to remove a dynamic network variable from a device's main interface. You cannot use the *Remove* method to remove a network variable from *NetworkVariables* collection on a device's main interface, even if it is a dynamic network variable. However, you can use the *MoveToInterface* method to move a dynamic network variable from the main interface to a custom interface. Once you have done so, you can remove the network variable from the custom interface, and its removal will be propagated to the main interface. The ability to move a dynamic *LonMarkObject* or network variable from one interface to another may also be useful after you have upgraded a device's interface with the *Upgrade* method. Some static network variables and network variables that existed on the old interface, but not the new one, will be converted to dynamic and stored in a custom *Interface* object created during the upgrade. You can use this method to move those objects back to their correct interface. If either the main interface, or an interface from another device, is specified as the new interface for the *LonMarkObject* or network variable, then the **LCA, #4 lcaErrInvalidOleObject** exception will be thrown. If you call this method on a static *LonMarkObject* or network

	variable, then the **LCA, #119 lcaErrInterfaceNotModifyable** exception will be thrown. You can determine if a *LonMarkObject* or network variable is dynamic by reading the object's *IsDynamic* property. If you attempt to call this method on a network variable that has been previously assigned to a network variable with the *AssignNetworkVariable* method, then the operation will fail, and the **NS, #164 lcaErrNsNvmtInUse** exception will be thrown, unless the network variable object the network variable has been assigned to is a member of the target interface specified as the *newInterface* element.
Availability	Local, full, and lightweight clients.
Syntax	*object.MoveToInterface newInterface* **Element** — **Description** *object* — The network variable to be acted upon. *newInterface* — The *Interface* object to which the object should be moved.
Added to API	LNS Release 3.20.

ToString

Summary	Creates a string description containing the identifying characteristics for a *NetworkVariable* object. You can use this method on a model network variable to create a string descriptor, which may be used to create a new network variable using the *AddNvFromString* method. Editing the string created from this method is not supported.
Availability	Local, full, and lightweight clients.
Syntax	*descString* = *nvObject*.**ToString**(*options)* **Element** — **Description** *descString* — A String containing the network variable description. *nvObject* — The *NetworkVariable* object to be acted on. *options* — This field is fixed at **0**, and it is reserved for future use.
Added to API	LNS Release 3.0.

Properties

The *NetworkVariable* object contains the following properties:

- *Aliases*
- *AppDevice*

- *AppDeviceName*
- *AuthenticationConfigFlag*
- *AuthenticationFlag*
- *ChangeableTypeSupport*
- *ClassId*
- *ConfigClassFlag*
- *ConfigProperties*
- *ConnectDescTemplate*
- *Description*
- *Direction*
- *DsFormatType*
- *DsIsDefaultFormat*
- *DsPollInterval*
- *DsPriority*
- *EstimatedMaxRate*
- *EstimatedRate*
- *Extensions*
- *FuncProfileDescription*
- *FuncProfileName*
- *FuncProfileProgrammaticName*
- *ImplementsCp*
- *Index*
- *IsConfigProperty*
- *IsDynamic*
- *IsPolled*
- *Length*
- *LmNumberManufacturerAssigned*
- *LonMarkMemberIndex*
- *LonMarkMemberNumber*
- *LonMarkObjectNumber*
- *MaxLength*
- *Name*
- *NVHubs*
- *NVTargets*
- *OfflineFlag*
- *Parent*
- *ParentInterface*
- *Priority*
- *PriorityConfigFlag*
- *ProgrammaticName*
- *Selector*
- *SelfDocumentation*
- *ServiceType*
- *ServiceTypeConfigFlag*
- *SnvtId*
- *SnvtTypeIsModifiable*
- *SyncFlag*
- *TypeSpec*

Aliases

Summary	Contains the *Aliases* collection object associated with the specified *NetworkVariable* object. The *Aliases* collection contains a collection of *Alias* objects, each one representing an alias used by the network variable. If the network variable is not using any aliases, the collection will be empty. If you try to access this property from a *NetworkVariable* object that is not associated with a device (for example, a *NetworkVariable* object fetched from a *DeviceTemplate* object), a **LCA, #38 LCA_APP_DEVICE_REQUIRED** exception will be generated. However, if you read the *Aliases* property from a *NetworkVariable* object whose parent *AppDevice* object was defined while in engineered mode (a *DeviceTemplate* was provided when the *AppDevice* was added), a valid *Aliases* collection object will be returned.
Availability	Local, full, and lightweight clients.
Syntax	*aliasesColl = nvObject.Aliases* **Element** — **Description** *aliasesColl* — The returned *Aliases* collection object. *nvObject* — The *NetworkVariable* object to be acted on.
Data Type	*Aliases* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

AppDevice

Summary	Identifies the *AppDevice* object containing this network variable.
Availability	Local, full, and lightweight clients.
Syntax	*appDevObject = object.AppDevice* **Element** — **Description** *appDevObject* — The *AppDevice* object to be returned. *object* — The *NetworkVariable* object to be acted on.
Data Type	*AppDevice* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

AppDeviceName

Summary	Returns the name of an application device that contains the network variable.
Availability	Local, full, and lightweight clients.
Syntax	*nvMtName* = *nvMtObject*.**AppDeviceName** **Element** — **Description** *nvMtObject* — The *NetworkVariable* object to be acted upon. *nvMtName* — The name of the network variable.
Data Type	String
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

AuthenticationConfigFlag

Summary	Defines whether the network variable's use of authentication may be changed in a connection.
Availability	Local, full, and lightweight clients.
Syntax	*authConfigFlag* = *nvObject*.**AuthenticationConfigFlag** **Element** — **Description** *authConfigFlag* — A Boolean value. **TRUE**. The use of authentication may be changed in a connection containing this network variable using the *ConnectDescTemplate* object. **FALSE**. The network variable's use of authentication may not be changed. *nvObject* — The *NetworkVariable* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

AuthenticationFlag

Summary	Returns the network variable's default authentication setting, which is specified by the device developer. If this property is set to True, the network variable uses authentication by default. You can override the default setting by writing to the *UseAuthenticationFlag* of the *ConnectDescTemplate* object used by the connections involving the network variable. **Note**: If you set the *UseAuthenticationFlag* property to True, then all devices participating in the authenticated connection must also have authentication enabled. This means that the *AuthenticationEnabled* property of each *AppDevice* involved in the connection must be set to True.
Availability	Local, full, and lightweight clients.
Syntax	*authFlag* = nvO*bject*.**AuthenticationFlag** **Element** — **Description** *authFlag* — A Boolean value indicating whether the network variable uses authentication by default. **TRUE**. The network variable uses authentication by default. **FALSE**. The network variable does not use authentication by default. *nvObject* — The *NetworkVariable* object being queried.
Data Type	Boolean
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ChangeableTypeSupport

Summary	Indicates whether you can change the type that this network variable uses.
Availability	Local, full, and lightweight clients.
Syntax	*changeable* = *networkVariable*.**ChangeableTypeSupport** **Element** — **Description** *changeable* — A Long value indicating whether the network variable's type can be changed. The possible values for this element, which are stored in the *ConstNvChangeableTypeSupport*

		constant, are as follows: **0 lcaNvChangeableTypeNone** The network variable's type cannot be changed. **1 lcaNvChangeableTypeSdOnly** You can change the network variable's type by writing to its *TypeSpec* or *SnvtId* property. When you write to this property, OpenLNS will change the type by modifying the self-documentation information of the device associated with the network variable. This is the value that will be returned by all dynamic network variables. You can determine if a network variable is dynamic by reading its *IsDynamic* property. **2 lcaNvChangeableTypeSCPT** You can change the network variable's type by writing to its *TypeSpec* or *SnvtId* property. When you write to this property, OpenLNS will change the type by modifying the **SCPTnvType** configuration property stored on the device associated with the network variable. The device's self-documentation information will remain unchanged. **Note**: If the value of the **SCPTnvType** configuration property used to modify the network variable type is unknown, or if the value of the configuration property is invalid (i.e. the type category is set to **NVT_CAT_NUL [-1]**), OpenLNS will set the self-documentation information of the device associated with the network variable to match the network variable's *SnvtId* stored in the OpenLNS database. You can set the *TypeSpec* property to the appropriate value to fix the following problems with the value of the **SCPTnvType** configuration property: • The configuration property's value is unknown because it was never set, or because its status was

	cleared, • The configuration property has been set to an invalid value explicitly by your application, or when your application downloaded or uploaded the default configuration property values with the *DownloadConfigProperties* or *UploadConfigProperties* methods. *networkVariable* The *NetworkVariable* object to be acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *NetworkVariable* object in the *ConstClassIds* constant: **22 lcaClassIdNetworkVariable** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ConfigClassFlag

Summary	Indicates whether the network variable is a configuration network variable (CPNV). A CPNV is used for application configuration. This should not be confused with a configuration property that is implemented as a network variable, and the related *IsConfigProperty* property. Network variables that implement configuration properties are always CPNVs, and the *ConfigClassFlag* and *IsConfigProperty* properties for these network variables are set to True. However, as described below, a network variable can be a CPNV without implementing a LonMark compliant configuration property. In this case, the *IsConfigProperty* is set to False. Device developers can create config network variables to implement configuration properties using the "**config_prop**" or "**cp**" keywords. This implements a configuration property by creating a CPNV with LONMARK compliant network variable strings. See the *NodeBuilder User's Guide* for more information on the "**config_prop**" or "**cp**" keywords. A CPNV may also be implemented using the "config" keyword. Typically these network variables will not implement configuration properties. However, legacy applications may also implement a configuration property by declaring a network variable with the "config" keyword, and hard-coding the appropriate LonMark compliant self-documentation string. See the *NodeBuilder User's Guide* for more information on the the "**config**" keyword.
Availability	Local, full, and lightweight clients.
Syntax	*configClassFlag* = *nvObject*.**ConfigClassFlag** **Element** — **Description** *configClassFlag* — A Boolean value indicating whether the network variable uses authentication by default. **TRUE**. The network variable is declared as a CPNV. **FALSE**. The network variable is not declared as a CPNV. *nvObject* — The *NetworkVariable* object being queried.
Data Type	Boolean
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConfigProperties

Summary	Contains the *ConfigProperties* collection object associated with the specified *NetworkVariable* object.
Availability	Local, full, and lightweight clients.
Syntax	*configPropsCollection* = *NetworkVariable.ConfigProperties* **Element** — **Description** *configPropsCollection* — The returned *ConfigProperties* collection. *NetworkVariable* — The *NetworkVariable* object to be acted on
Data Type	*ConfigProperties* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ConnectDescTemplate

Summary	Contains the *ConnectDescTemplate* object associated with the specified *NetworkVariable* object. This template contains parameters used when creating connections. Setting the *ConnectDescTemplate* property with a modified *ConnectDescTemplate* object will cause the attributes of a connection to be updated, if the network variable the property belongs to is the hub for that connection. If the network variable is not yet added to a connection, it is assumed that you will call the *Connect* method to create the connection immediately after setting this property. If you do not, reading the *ConnectDescTemplate* property will return the old (default) *ConnectDescTemplate* object. To modify the attributes of an existing connection description, modify its *ConnectDescTemplate* properties as if it were a new object. Setting the *ConnectDescTemplate* property with a modified *ConnectDescTemplate* object will cause the attributes of a connection to be updated, if the network variable the *ConnectDescTemplate* object belongs to is the hub for that connection.
Availability	Local, full, and lightweight clients.
Syntax	*cdTemplateObject* = *object.ConnectDescTemplate* **Element** — **Description** *cdTemplateObject* — The *ConnectDescTemplate* object. *object* — The *NetworkVariable* object.
Data Type	*ConnectDescTemplate* object.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *NetworkVariable* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *NetworkVariable* object . *object* — The *NetworkVariable* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Direction

Summary	Specifies whether the network variable is an input or output network variable.
Availability	Local, full, and lightweight clients.
Syntax	*directionValue* = *object*.**Direction** **Element** — **Description** *directionValue* — The direction of the network variable. The vallid values for this element, which are contained in the *ConstNvDirections* constant, are as follows: **0 lcaNvDirectionInput** The network variable is an input network variable. **1 lcaNvDirectionOutput** The network variable is an output network variable. *object* — The *NetworkVariable* object.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DsFormatType

Summary	Specifies the type the OpenLNS Server will use in interpreting the value of the network variable.
Availability	Local, full, and lightweight clients.
Syntax	*formatType* = *nvObject*.**DsFormatType** **Element** — **Description** *formatType* — The type the OpenLNS Server will use in interpreting the value of the network variable. The *DsFormatType* parameter is a string defining the type that will be used to interpret the value of the network variable. The *DsFormatType* string has several possible syntaxes: 1. A format name beginning with "SNVT" for standard network variable type formats, or "UNVT" for user-defined network variable type formats. For example, "SNVT_switch". The OpenLNS Server will use the resource file catalog specified using the *LdrfCatalogPath* property to search for SNVT and UNVT formats. There may be several different formats for the same network variable type. For example, the LonMark standard device resource file set has two alternate formats for the type `SNVT_temp_f`, and those formats are named `SNVT_temp_f#SI` and `SNVT_temp_f#US`. In this case, you could specify SNVT_temp_f to get the current default format for that type. The default formats for types such as SNVT_temp_f are determined using the Windows regional settings of the PC where the data is formatted (on the PC housing the OpenLNS Server for local and lightweight clients, or on the PC running the application for full clients). To determine the settings being used on a computer, open the Windows Control Panel and double-click the **Regional Options** icon. Select the **Numbers** tab.

	OpenLNS uses the value of the Measurement System field (either U.S. or metric) on this tab to determine the default format to use for types such as SNVT_temp_f. You can also specify a full format name (i.e. `SNVT_temp_f#SI` for or `SNVT_temp_f#US)` to select a specific format for that type. For UNVTs, you must always specify a fully-qualified format name. 2. In some cases, a format exists for a given SNVT or UNVT that has the same name as the underlying type. This is called the root format for the type, and it may be different from the default format for that type. In order to explicitly use the root format, you will need to append a '#' character to the format name you write to this property to indicate that you are specifying the format name (and not the type name) for this type. For example, if you read the value of a network variable of type SNVT_time_stamp when the *CategoryPreferenceList* property of the *FormatLocale* object your application is using is set to LO, and you set the DsFormatType property to SNVT_time_stamp, then the data stored in the *Value* property will be formatted using the SNVT_time_stamp #LO format. However, if you set the DsFormatType property to SNVT_time_stamp#, the data stored in the *Value* property will be formatted using the root SNVT_time_stamp format. 3. A fully-qualified format name, expressed in the following syntax: `"#<progID>[<scope>].<format name>"` In this syntax, the "#", "[", "]" and "." characters are literal characters. A hex byte string represents the program ID. The scope is a one-digit string. It represents a filter that indicates relevant parts

	of the program ID, and may be one of the following: **0 -** Standard **1** - Device Class **2** - Device Class and Usage **3** - Manufacturer **4** - Manufacturer and Device Class **5** - Manufacturer, Device Class, and Device Subclass **6** - Manufacturer, Device Class, Device Subclass, and Device Model For example: #800001128000000[4].UNVT_date_event 4. The name of one of the built-in types used by the OpenLNS Server, which include "INT", "REAL", "DISCRETE", "BINARY", "RAW" and "RAW_HEX". Alternatively, you can use the values defined in the *ConstDsFormatTypes* constant, which are as follows: **"BINARY"** lcaDsFormatTypeBinary **"DISCRETE"** lcaDsFormatTypeDiscrete **"REAL"** lcaDsFormatTypeFloat **"INT"** lcaDsFormatTypeInteger **"RAW"** lcaDsFormatTypeRaw **"RAW_HEX"** lcaDsFormatTypeRawHex **"RAW_HEX_PACKED** "lcaDsFormatTypeRawHexPacked **"STRING"** lcaDsFormatTypeString If you attempt to write to this property and the setting fails for any reason, the network variable will retain the last type assigned to it. The setting may fail if you assign an invalid type string to this property, or if the *LdrfCatalogPath* property does not contain the

		correct path to the LonMark Device Resource File catalog. After you write to this property, you need to invoke the *DsSaveOptions* method to save the new value into the OpenLNS database. You can revert this property to the network variable's default type by writing an empty string to this property. You can check if the network variable is currently using its default type by reading the *DsIsDefaultFormat* property. If you modify the value of the *DsFormatType* property and the resource file defining the network variable's new type becomes unavailable to the OpenLNS Object Server, the value of the *DsFormatType* property will not change. However, when you attempt to read the value of the network variable, the **DS, #60 lcaErrDsTypeFileNotFound** exception will be thrown. If you then create a data point to read or write to the value of the network variable, the data point's *DsIsDefaultFormat* property will be displayed using the type assigned to the *FallbackFormat* property of the *FormatLocale* object your application is using. Note that for Local and Full client applications, the value of a network variable is formatted on the client computer. For Lightweight client applications, the value is formatted on the OpenLNS Server computer. In either case, the *DsFormatType* property indicates which type the OpenLNS Server will use to format the value of the network variable. This property applies only to points monitored and controlled using single-point monitoring. If you are using monitor set monitoring, use the *FormatSpec* property. **Note:** The old syntax of <file name>.<type name> for DsFormatType is no longer supported. This notation

	was used for pre-LonMark resource files. *nvObject* The *NetworkVariable* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

DsIsDefaultFormat

Summary	Indicates whether the network variable is using its default format. OpenLNS determines the default format to use for SNVT network variables by reading the SNVT ID of the network variable, and using it to locate the applicable entry for that SNVT type in the standard resource files. OpenLNS then reads the default type from the resource file. OpenLNS determines the default type to use for UNVT network variables on LonMark objects by reading the Functional Profile Template of the LonMark object and locating the applicable entry for the UNVT type in the user-defined resource files, based on the NV members contained within the FPT. Once this is done, it reads the default type for the UNVT from the resource file entry. UNVT network variables that are not on LonMark objects default to the type "RAW." If OpenLNS cannot read the format files, or if it is unable to determine a network variable's default format for any reason, the format assigned to the network variable will be "RAW." The *LdrfCatalogPath* of the *System* object must point to the actual location of the LonMark Device Resource File catalog for OpenLNS to read the format files. You can change the format assigned to a network variable by writing to its *DsFormatType* property. You can restore the default format to a network variable at any time by writing an empty string to the *DsFormatType* property. Once you have done so, the *DsIsDefaultFormat* property will return True.
Availability	Local, full, and lightweight clients.
Syntax	*isDefault* = *nvObject.DsIsDefaultFormat* **Element** **Description** *isDefault* A Boolean value, which depends on the value returned by the *DsFormatType* property. **TRUE**. The *DsFormatType* property is the same as the network

	variable's type, or it is set to the default format for the network variable given the *locale settings* currently in effect. **FALSE**. The *DsFormatType* property is not the same as the network variable's type. For example, if the current locale settings are set to use US measurement units, then for a network variable of type **SNVT_temp_f**, the *DsIsDefaultFormat* property will return True if the *DsFormatType* property is set to **SNVT_temp_f** or **SNVT_temp_f#US**. *nvObject* The *NetworkVariable* object to be acted on.
Data Type	Boolean.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

DsPollInterval

Summary	Specifies the poll or throttle interval (in tenths of a second) used by the Data Server. For remote clients using the shared Data Server mode, this property specifies the batch update rate. The *DsPollInterval* property specifies either a poll or throttle interval, depending on the context in which it is used. • For polled network variables, the property specifies the interval between polls. Setting the value to **0** disables polling. • For bound network variables, the property specifies the minimum interval which must elapse before the Data Server will generate an update event. The throttling interval may be used to regulate the rate in which *OnNetworkVariableUpdate* events occur. Remote client applications that use shared access mode receive batched data updates. Instead of receiving individual network variable value updates over the IP network, the client receives a regular update containing the updated values for the monitored network variables. In this scenario, the *System* object's *DsPollInterval* property specifies the batch update rate for each network variable. If you are developing a remote application that uses shared access mode (meaning that the DsMode property is set to **lcaDsModeShared**), you cannot set the *NetworkVariable*

<table>
<tr><td></td><td>object’s DsPollInterval property. An exception will be raised if you attempt to do so.

The System object's DsPollInterval property sets the default value which is applied when a NetworkVariable object's DsPollInterval property is left unspecified. Once an application has explicitly set the value of the DsPollInterval property of the System object, it will not be reverted back to the default value by OpenLNS, and must be maintained manually from that point on.

Invoke the DsSaveOptions to save the property value into the persistent database.

This property applies only to points monitored and controlled using single-point monitoring. If you are using monitor set monitoring, use the PollInterval property.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>intervalValue = object.DsPollInterval

Element — Description

intervalValue — Poll or throttle interval, in tenths of a second. The allowed range is 0 to 33554431 (0x01FFFFFF). This maximum value corresponds to a time of approximately 38.8 days. The default value is 10 (one second) for System .DsPollInterval.

The system's current DsPollInterval value establishes the default that will be used by newly created NetworkVariable objects whose DsPollInterval value is left unspecified.

object — The NetworkVariable object to be acted on.</td></tr>
<tr><td>Data Type</td><td>Integer.</td></tr>
<tr><td>Read/Write</td><td>Read/write.</td></tr>
<tr><td>Added to API</td><td>Prior to LNS Release 3.0.</td></tr>
</table>

DsPriority

Summary	Specifies whether the Data Server polls or updates a network variable using priority messages. **Note**: This property is deprecated because it was only useful with single-point monitoring. You should use temporary monitor sets instead of single-point monitoring. When doing so, you can use the *Priority* property to determine the priority assigned to each message. For more information on temporary monitor sets, see the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*priorityFlag* = *object*.**DsPriority** **Element** — **Description** *priorityFlag* — The priority flag value. **TRUE**. The Data Server uses priority messaging to update a network variable. **FALSE**. The Data Server polls a network variable. This is the default. *object* — The *NetworkVariable* object to be acted on.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

EstimatedMaxRate

Summary	The estimated maximum message rate (in tenths of messages per second) declared for the network variable.
Availability	Local, full, and lightweight clients.
Syntax	*estMaxRateValue* = *nvObject*.**EstimatedMaxRate** **Element** — **Description** *estMaxRateValue* — Estimated sustained message rate. The valid values for this property are integers from 0 to 18780 (0 to 1878.0 messages per second). *nvObject* — The *NetworkVariable* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

EstimatedRate

Summary	The estimated sustained message rate (in tenths of messages per second) declared for the network variable.
Availability	Local, full, and lightweight clients.
Syntax	*estimatedRateValue* = *nvObject*.**EstimatedRate** **Element** — **Description** *estimatedRateValue* — Estimated sustained message rate. The valid values for this property are integers from 0 to 18780 (0 to 1878.0 messages per second). *nvObject* — The *NetworkVariable* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Extensions

Summary	Retrieves a child *Extensions* collection from a parent *NetworkVariable* object.
Availability	Local, full, and lightweight clients.
Syntax	*extensions* = *networkVariable*.*Extensions***()** **Element** — **Description** *networkVariable* — The parent *NetworkVariable* object. *extensions* — The child *Extensions* collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

FuncProfileDescription

Summary	Provides a desciptive comment of the functional profile associated with the *NetworkVariable* object. This property is accessed from the functional profile template file associated with the object. This property is language dependent. Set the *System* object's *ResourceLanguageId* to control the language.
Availability	Local, full, and lightweight clients.
Syntax	*description* = *object*.**FuncProfileDescription** **Element** — **Description**

	object	The *NetworkVariable* object to be acted on.
	description	The returned functional profile description string.
Data Type	String.	
Read/Write	Read-only.	
Added to API	Prior to LNS Release 3.0.	

FuncProfileName

Summary	Returns the functional profile name associated with the *NetworkVariable* object. This property is accessed from the functional profile template file associated with the object. The name returned by this property is accessed from the functional profile template file associated with this object. This property is language dependent. Set the *System* object's *ResourceLanguageId* to control the language.	
Availability	Local, full, and lightweight clients.	
Syntax	*typeNameValue* = *object*.**FuncProfileName**	
	Element	**Description**
	object	The *NetworkVariable* object to be acted on.
	typeNameValue	The functional profile name to be returned.
Data Type	String.	
Read/Write	Read-only.	
Added to API	Prior to LNS Release 3.0.	

FuncProfileProgrammaticName

Summary	Returns the functional profile programmatic name associated with the *NetworkVariable* object. This name is accessed from the functional profile template file associated with the object. The programmatic name is the base name stored for the object; it is not language dependent like the *FuncProfileName* property.	
Availability	Local, full, and lightweight clients.	
Syntax	*progNameValue* = *Object*.**FuncProfileProgrammaticName**	
	Element	**Description**
	progName	The functional profile programmatic name of the object.
	Object	The *NetworkVariable* object to be acted

	on.
Data Type	String.
Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

ImplementsCp

Summary	If the network variable is a configuration network variable, returns the configuration property object (possibly a shared instance) implemented by this network variable. If this network variable is not a configuration network variable, then accessing this property throws an **LCA, #166** error *("network variable does not implement a configuration property").* To determine whether this network variable is a configuration network variable, read the *IsConfigProperty* property. A configuration network variable implements a single configuration property, but each configuration property can apply to multiple objects at the same level (device/template, LonMark object, or network variable). To determine which objects this configuration network variable affects, read the *AppliesTo* property of the *ConfigProperty* object. The network variable-based configuration property returned by this property will refer back to this implementing configuration network variable through the *ConfigNv* property of the *ConfigProperty* object.
Availability	Local, full, and lightweight clients.
Syntax	*cpObject* = *nvObject*.**ImplementsCp** **Element** — **Description** *cpObject* — The *ConfigProperty* object implemented by this network variable. *nvObject* — The configuration network variable to be acted on.
Data Type	*ConfigProperty* object.
Read/Write	Read only.
Added to API	OpenLNS.

Index

Summary	Returns the index within an application device of the network variable.
Availability	Local, full, and lightweight clients.

Syntax	*index* = *object*.**Index** **Element** **Description** *object* The object to be acted on. *index* Index of the network variable. The valid values for this element are 0–14.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IsConfigProperty

Summary	Indicates whether the network variable implements a configuration property that conforms to the LonMark interoperability guidelines. A True value indicates that the network variable implements a configuration property conforming to LonMark interoperability guidelines. The related *ConfigClassFlag* property will be set to True whenever the *IsConfigProperty* property is set to True. However, as described below, some network variables may have the *ConfigClassFlag* property set to True, and still have the *IsConfigProperty* property set to False. Currently, there are two ways in which a device developer can implement a configuration property as a network variable: 1. Use the "**config_prop**" or "**cp**" keywords, which is the recommended, newer method. This method automatically creates a configuration network variable (CPNV) with an appropriate SelfDocumentation string that specifies the configuration property attributes. See the *NodeBuilder User's Guide* for more information on the "**config_prop**" or "**cp**" keywords. 2. Use the "**config**" keyword. Any network variable declared with the "**config**" keyword will have a *ConfigClassFlag* value of True. However, in order to implement a configuration property (and therefore have the *IsConfigProperty* set to True), you must manually provide a LonMark compliant SelfDocumentation string that specify the attributes of the configuration property. See the *NodeBuilder User's Guide* for more information on the "**config**" keyword.
Availability	Local, full, and lightweight clients.
Syntax	*cpFlagValue* = *nvObject.IsConfigProperty* **Element** **Description** *cpFlagValue* A Boolean value. **TRUE**. The network variable

		implements a configuration property conforming to LonMark interoperability guidelines. **FALSE**. The network variable does not implement a configuration property. The *NetworkVariable* object to be acted on
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

IsDynamic

Summary	Indicates whether the network variable is dynamic. A network variable is considered to be dynamic if it was manually added to the interface it belongs to, rather than being part of the static interface defined by a device template.	
Availability	Local, full, and lightweight clients.	
Syntax	*isDynamicFlag = object.IsDynamic*	
	Element	**Description**
	isDynamicFlag	A True of False value indicating whether the object is dynamic. **TRUE**. The network variable is dynamic. **FALSE**. The network variable is static.
	object	The *NetworkVariable* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

IsPolled

Summary	Indicates whether the network variable has the poll attribute enabled.	
Availability	Local, full, and lightweight clients.	
Syntax	*isPolledFlag = nvObject.***IsPolled**	
	Element	**Description**
	isPolledFlag	A Boolean value. This element may be one of the following values:

		1 The network variable's poll attribute on. If the *Direction* property is set to **lcaNvDirectionOutput**, this network variable will not send network variable update messages automatically; otherwise, this network variable may poll other network variables. **0** The network variable's poll attribute off. If the Direction property is set to **lcaNvDirectionOutput**, this network variable sends update messages automatically; otherwise, this network variable does not poll other network variables.
	nvObject	The *NetworkVariable* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

Length

Summary	Contains the length (in bytes) of the network variable.	
Availability	Local, full, and lightweight clients.	
Syntax	*lengthValue* = *nvObject*.**Length**	
	Element	**Description**
	lengthValue	The length of the network variable in bytes. A value of 0 indicates that the length cannot be determined by the Object Server. If the network variable supports changeable types, you can use its *TypeSpec* property to change its type. In doing so, you can alter the length of the network variable
	nvObject	The *NetworkVariable* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

LmNumberManufacturerAssigned

Summary	Indicates whether the member number within a LonMark object was assigned by the device manufacturer instead of being specified in the *LonMark Application Layer Interoperability Guidelines* or in a LonMark approved functional profile.
Availability	Local, full, and lightweight clients.
Syntax	*lmNumberFlag* = *nvObject*.**LmNumberManufacturerAssigned** **Element** — **Description** *lmNumberFlag* — A Boolean value indicating whether the member number is manufacturer assigned. **TRUE**. The member number was assigned by the device manufacturer. **FALSE**. The member number was specified in the *LonMark Application Layer Interoperability Guidelines* or in a LonMark approved functional profile. *nvObject* — The *NetworkVariable* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LonMarkMemberIndex

Summary	Contains the ordinal index of the network variable within a *LonMarkObject* object. Each *LonMarkObject* object contains some number of network variables. Each of those network variables is assigned a member index within that *LonMarkObject* based on its position within the object. This property differs from the *LonMarkMemberNumber* property, which contains the member number assigned to the network variable by the *LonMark Application Layer Interoperability* guidelines or by the user (in the case of a network variable member not specified in those guidelines).
Availability	Local, full, and lightweight clients.

Syntax	*indexValue* = *nvObject*.**LonMarkMemberIndex** **Element** — **Description** *indexValue* — The index value of the network variable as a long. On static interfaces, indexes are numbered from 0 to *n*-1, where *n* equals the number of network variables in the object. If dynamic network variables have been assigned and subsequently removed from a *LonMarkObject*, the member indexes may not be contigious, and some member indexes may be equal to or greater than the number of network variables in the object. If the network variable does not belong to a *LonMarkObject*, this property will contain the value –1. *nvObject* — The *NetworkVariable* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LonMarkMemberNumber

Summary	Contains the member number assigned to this *NetworkVariable* within a *LonMarkObject*. Each network variable contained within a *LonMarkObject* is assigned a member number which, when combined with the *LmNumberManufacturerAssigned* property, is unique within the object. Member numbers are 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. If the network variable does not belong to a *LonMarkObject*, this property will contain the value –1. This property differs from the *LonMarkMemberIndex* property, which contains the index number assigned to the network variable within the *LonMarkObject*. This generally ranges from 0 to *n*-1 (on a static interface, it always reanges from 0..*n*-1), where *n* represents the number of network variables assigned to the *LonMarkObject*.
Availability	Local, full, and lightweight clients.

Syntax	*memNumValue* = *nvObject*.**LonMarkMemberNumber** **Element** **Description** *memNumValue* The member number of the network variable as a Long. *nvObject* The *NetworkVariable* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LonMarkObjectNumber

Summary	Contains the object number of the *LonMarkObject* to which this network variable is assigned. Objects on a device are generally numbered from 0 to *n*-1, where *n* is the number of objects on the device. This is always the case on devices with static interfaces. If a device supports dynamic function blocks, some *LonMarkObject* objects may be assigned an object number greater than or equal to the value *n* (the number of objects on the device). If the network variable does not belong to a *LonMarkObject*, this property will return –1. You can use the *LonMarkObjects* collection's *ItemByIndex* method to access the *LonMarkObject* containing the network variable. You will need to reference the *LonMarkObject* by its device index when you call the method.
Availability	Local, full, and lightweight clients.
Syntax	*objNumValue* = *nvObject*.**LonMarkObjectNumber** **Element** **Description** *objNumValue* The object number of the network variable as a Long. *nvObject* The *NetworkVariable* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MaxLength

Summary	Contains the maximum length (in bytes) of the network variable. You can determine the actual length of a network variable by reading the *Length* property. If the network variable does not

	support changeable types, then the value of the *MaxLength* property will always match the value of the *Length* property. You can determine if a network variable supports changeable types by reading the network variable's *ChangeableTypeSupport* property. If the device supports changeable types, and the configuration properties are avaliable, then this property will be set to the maximum network variable length as defined by **SCPTmaxNvLength** configuration property reported by the device for that network variable. In this case, reading this property is the same as reading that configuration property directly. It may result in an attempt to read the value from the device, if the configuration property value is unknown to OpenLNS. You can determine if the configuration properties for a device are available by reading the *ConfigPropertiesAvailable* property of the *Interface* object used by the device. If the network variable supports changeable types and the configuration properties are not available, or the device does not contain a **SCPTmaxNvLength** configuration property for the network variable, this property will be set to –1.
Availability	Local, full, and lightweight clients.
Syntax	*maximum* = *networkVariable*.**MaxLength** **Element** — **Description** *maximum* — The maximum length of the network variable, in bytes. *networkVariable* — The *NetworkVariable* object to be acted upon.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name**

	Element	**Description**
	stringValue	The name of the object.
	object	The object to be acted on.
Data Type	String.	
Read/Write	Read/write.	
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.	

NvHubs

Summary	Returns a collection containing all network variable hubs for which the network variable is a target, including the network variable if it is the hub for a connection. In conjunction with the *NVTargets* property, this property enables you to manage complex network connections involving multiple hubs and sets of targets. When a new hub is added, it will not neccessarily be added to the end of the list of hubs; therefore, you should update the cached copy of the complete hub list when you add or delete a hub.	
Availability	Local, full, and lightweight clients.	
Syntax	*nvCollection* = *nvObject*.**NvHubs**	
	Element	**Description**
	nvCollection	The returned *NetworkVariables* collection.
	nvObject	The specified *NetworkVariable* object.
Data Type	*NetworkVariables* collection object.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

NvTargets

Summary	Contains the target network variables for a hub. If the network variable is not the hub for a connection, the *NVTargets* property will return an empty collection object.	
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.	
Syntax	*targetCollection* = nvO*bject.NVTargets*	
	Element	**Description**
	targetCollection	The collection of target *NetworkVariable* objects.

	nvObject The specified *NetworkVariable* object.
Data Type	*NetworkVariables* collection object.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

OfflineFlag

Summary	Indicates whether the network variable should be updated only while the device's *State* is set to **lcaStateSoftOffline**.
Availability	Local, full, and lightweight clients.
Syntax	*offlineFlagValue* = *nvObject*.**OfflineFlag** **Element** **Description** *offlineFlagValue* A Boolean value indicating whether the member number is manufacturer assigned. **TRUE**. The network variable should be updated only while the device's *State* is set to **lcaStateSoftOffline**. **FALSE**. The network variable may be updated while the device's *State* is set to other states. *nvObject* The *NetworkVariable* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is

	desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ParentInterface

Summary	Returns the *Interface* object to which the *NetworkVariable* Object belongs. Each *AppDevice* on a network contains an *Interface* property containing the device's main interface, and an *Interfaces* collection containing the custom interfaces that have been added to the device dynamically. The interfaces each contain network variables and LonMarkObjects that reflect the device's functionality on the network. The *ParentInterface* property returns the *Interface* object to which the *NetworkVariable* Object belongs. In the case of static LonMarkObjects and network variables, this property returns the main device interface or device template the object is associated with. In the case of a dynamic *LonMarkObject* and network variables, it returns the custom interface to which the object belongs. The *ParentInterface* property is read-only. However, you can move a dynamic network variable or *LonMarkObject* from one custom interface to another with the method.
Availability	Local, full, and lightweight clients.
Syntax	*interface* = *object*.**ParentInterface** **Element** — **Description** *interface* — The *Interface* object returned by the property. *object* — The *NetworkVariable* object to be acted upon.
Data Type	*Interface* object.
Read/Write	Read only.

Priority

Summary	Contains the default priority setting for the network variable, as specified by the device developer. If the network variable is a connected output network variable, or a polling input network variable, then this flag will be used to determine whether messages sent by the network variable use priority. This value may be overridden by the *UsePriorityFlag* property of the network variable's

	ConnectDescTemplate object.
Availability	Local, full, and lightweight clients.
Syntax	*priorityValue* = o*bject.Priority* **Element** **Description** *priorityValue* The default priority setting assigned to the network variable. *object* The *AppDevice* object to be acted on.
Data Type	Integer.
Read/Write	Read only
Added to API	Prior to LNS Release 3.0.

PriorityConfigFlag

Summary	Indicates whether priority is configurable for the network variable.
Availability	Local, full, and lightweight clients.
Syntax	*pcFlagValue* = *nvObject*.**PriorityConfigFlag** **Element** **Description** *pcFlagValue* A True of False value indicating whether the object is dynamic. **TRUE**. The network variable's use of priority may be changed in a connection via the *ConnectDescTemplate* object. **FALSE**. The network variable's use of priority may not be changed. *nvObject* The *Network* variable object.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ProgrammaticName

Summary	Returns the programmatic name of the network variable. This property contains the orignal base name of the network variable as "programmed" in the device containing the object. • For static network variables, OpenLNS initially acquires the name from the functional profile template file or external interface associated with the device. • For dynamic network variables, you will specify the name to use when you create the object with the appropriate *Add* method. Initially, the *Name* property, which represents the user name of the network variable or network variable, will also be set to the same value as the *ProgrammaticName* property. The *Name* property can be subsequently changed for all *network variable*s, allowing OpenLNS applications to create their own identifying names for those objects. This property can be a maximum of 16 characters long, and must conform to the character restrictions defined in version 3.3 and later of the *LonMark Application Layer Interoperability Guidelines*. You may not use the following characters in the *ProgrammaticName* property: the forward slash (/), back slash (\), period (.), and colon (:).
Availability	Local, full, and lightweight clients.
Syntax	*progName = Object.ProgrammaticName* **Element** — **Description** *progName* — The *ProgrammaticName* of the object. *Object* — The network variable to be acted on.
Data Type	String.
Read/Write	Read-only for static network variables. Read/write for dynamic network variables. You can check if a network variable is dynamic or static by reading the *IsDynamic* property. When writing to this property, you should note that some devices, such as the SmartServer, require that all network variables within the device have a unique programmatic name. In addition, all network variables contained within a custom *Interface* object must have unique programmatic names. If you attempt to assign a duplicate programmatic name to a network variable on such a device, the operation will fail, and the **LCA, #132 lcaErrUniqueNvNameRequired** exception will be thrown.
Added to API	Prior to LNS Release 3.0.

Selector

Summary	The network variable selector value assigned to this network variable alias. When a device is installed, selector values that represent unbound network variables are assigned to the network variables in that device. When placing the network variable in a connection, the OpenLNS Object Server assigns a value representing that connection.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*selectorValue* = *object*.**Selector** **Element** — **Description** *object* — The *NetworkVariable* object to be acted on. *selectorValue* — The network variable selector value
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

SelfDocumentation

Summary	Stores the self-documentation string of the network variable. The length of the string is not provided as a separate property. To get the length, get the *descriptionString*, and calculate the length from it. Note that this property returns only the user portion (which follows the LonMark portion, if any) of the self-documentation string
Availability	Local, full, and lightweight clients.
Syntax	*descriptionString* = *object*.**SelfDocumentation** **Element** — **Description** *programIdValue* — The program ID value of the object. *object* — The *NetworkVariable* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ServiceType

Summary	Stores the messaging service to be used for the connection. **Note**: This property will only be read by the OpenLNS Object Server if the **lcaConnPropsServiceType** option is set in the *ConnectDescTemplate* object's *PropertyOptions* property. If the **lcaConnPropServiceType** option is not set, then OpenLNS Object Server will determine the service type for connections using the *ConnectDescTemplate* object by reading the ServiceType property of the connection's hub network variable.
Availability	Local, full, and lightweight clients.
Syntax	*serviceTypeValue* = *object*.**ServiceType** **Element** — **Description** *serviceTypeValue* — The service type to be used. The enumerated values for this element, which are contained in the *ConstServiceTypes* constant, are as follows: **0 lcaSvcAckd** Acknowledged messaging service. The device sends an acknowledgment message after it has received the message. If your application will be sending messages to large numbers of devices at once, one of the unacknowledged messaging services may be desirable, as the acknowledgment messages may generate a significant amount of network traffic. **1 lcaSvcUnackdRpt** Unacknowledged repeat messaging service. The device does not send acknowledgment messages; however, repeat messages are sent to the device after the initial message is sent to it to ensure that it reaches its destination. You can set the number of repeat messages to send, and the interval at which they will be sent, by writing to the *RepeatCount* and *RepeatTimer* properties. **2 lcaSvcUnackd** Unacknowledged messaging service. The device does not send

		acknowledgment messages. Do not use this service type on channels that support altnerate frequencies because the message will only be sent using the primary path. See the *AltPathType* property for more information. **3 lcaSvcRequest** Request/Response messaging service. You can use this value when sending explicit messages if the device receiving the message is designed to send a response message for the specified message code.
	object	The *NetworkVariable* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

ServiceTypeConfigFlag

Summary	Indicates whether the *ServiceType* is configurable.	
Availability	Local, full, and lightweight clients.	
Syntax	*flagValue* = *nvObject*.**ServiceTypeConfigFlag**	
	Element	**Description**
	flagValue	A Boolean value indicating whether the *ServiceType* property can be changed using the *ConnectDescTemplate* object. **TRUE**. The *ServiceType* property may be changed using *ConnectDescTemplate* object. **FALSE**. The *ServiceType* property may not be changed using *ConnectDescTemplate* object.
	nvObject	The *NetworkVariable* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

SnvtId

Summary	Contains the ID of the standard network variable type (SNVT) used by the network variable. If the network variable is using a user-defined type (UNVT), this property returns the value **0**. For more information on SNVTs and their IDs, go to the LONMARK Web site at *www.lonmark.org*. If the device supports modifiable types, this property may be modified. This is useful for devices that support flexible hardware options, and must have their network variable types modified to match the attached hardware. You can determine if the network variable supports modifiable types by reading the *ChangeableTypeSupport* or *SnvtTypeIsModifiable* properties. **Note**: Use the *TypeSpec* property to change the type and format used by a network variable.
Availability	Local, full, and lightweight clients.
Syntax	*snvtType* = *nvObject.SnvtId* **Element** — **Description** *snvtType* — The SNVT ID of the network variable, or **0** if the network variable is a UNVT *nvObject* — The *NetworkVariable* object to be acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

SnvtTypeIsModifiable

Summary	Indicates whether the Standard Network Variable Type (SNVT) ID of the network variable can be changed. Changing a network variables's type is useful for devices that support flexible hardware options, and must have their network variable types modified to match the attached hardware. You can use the network variable's *TypeSpec* property to change the type and format used by a network variable. Note that if a network variable is not using a SNVT type, you can determine if its type can be modified by reading the *ChangeableTypeSupport* property. The *ChangeableTypeSupport* property also returns information indicating whether or not you can change and network variable's type, and how OpenLNS would change the type (by modifying the self-documentation information of the device associated with the network variable, or by modifying the **SCPTnvType** configuration property stored on the

	device associated with the network variable).
Availability	Local, full, and lightweight clients.
Syntax	*snvtFlag* = *nvObject*.**SnvtTypeIsModifiable** **Element** — **Description** *snvtFlag* — A Boolean value indicating whether the network variable's type can be changed. **TRUE**. The *SNVT type* assigned to the network variable can be changed. **FALSE**. The *SNVT type* property may not be changed. *nvObject* — *NetworkVariable* object.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

SyncFlag

Summary	Indicates whether the network variable is a synchronous network variable. For synchronous network variables, all updates are delivered to the application program, not just the most recently received update (when earlier updates have been queued for but not delivered to the application program).
Availability	Local, full, and lightweight clients.
Syntax	*flagValue* = *nvObject*.**SyncFlag** **Element** — **Description** *flagValue* — A Boolean value indicating whether the network variable is a synchronous network variable. **TRUE**. The network variable is a synchronous network variable. **FALSE**. The network variable is not synchronous. *nvObject* — *NetworkVariable* object.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TypeSpec

Summary	Contains an object that defines the type used by the network variable. This includes the name of the network variable type, and the scope and program ID of the resource file that contains the definition of the type. OpenLNS uses this information to identify the type a network variable should use. In some cases, you can write to the properties of this object to change the network variable's type. If the *ChangeableTypeSupport* property of the network variable is set to **lcaNvChangeableTypeSdOnly (1)** or **lcaNvChangeableTypeSCPT (2)**, you can modify the network variable's type by writing new values to the properties of the *TypeSpec* object. For more information on this, see the *TypeSpec* object. If the *ChangeableTypeSupport* property is set to **lcaNvChangeableTypeNone (0)**, then you cannot modify the network variable's type. If you attempt to write to this property, the **LCA#:157 lcaErrTypeNotChangeable** exception will be thrown. The *TypeSpec* object contained within this property is not passed by reference. If you modify the values assigned to the properties of a local *TypeSpec* object, you must then explicitly assign the modified *TypeSpec* object back to the *TypeSpec* property of the *NetworkVariable* for the changes to take effect. See the *TypeSpec* object for more information.
Availability	Local, full, and lightweight clients.
Syntax	*typeSpecObject* = *networkVariable*.**TypeSpec** **Element** — **Description** *typeSpecObject* — The *TypeSpec* object returned by the property. *networkVariable* — The *NetworkVariable* object to be acted upon.
Data Type	*TypeSpec* object.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

NetworkVariables

A *NetworkVariables* object is a collection of *NetworkVariable* objects. You can use the *Item* property, the *ItemByProgrammaticName* method, or the *ItemByIndex* method to retrieve a network variable from the collection. The following table summarizes the *NetworkVariables* object.

Description	A collection of *NetworkVariable* objects.
Added to API	Prior to LNS Release 3.0.

Accessed Through	*Connections* object. *Interface* object. *LonMarkObject* object.
Default Property	*Item*
Methods	• *Add* • *ItemByIndex* • *ItemByProgammaticName* • *Remove* • *RemoveByIndex*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *NetworkVariables* object contains the following methods.

- *Add*
- *ItemByIndex*
- *ItemByProgammaticName*
- *Remove*
- *RemoveByIndex*

Add

Summary	Adds a network variable to the collection. You can use this on *NetworkVariables* collections that are contained by dynamic *LonMarkObjects*, or by custom *Interface* objects. The *Name* and *ProgrammaticName* properties of the *NetworkVariable* object will be set to match the specified *nvName* element when this method is called. The name can be a maximum of 16 characters, and it must not start with a number. In addition, it cannot include square brackets ([]), commas (,), or periods (.). Network variables contained within the same custom Interface object must have unique user names (*Name* property). In addition, some devices, such as the SmartServer, require that all network variables within the device have a unique programmatic name (*ProgrammaticName* property). If you attempt to assign a duplicate user name or programmatic name to a network variable on such a device or interface, the operation will fail, and the **LCA, #132 lcaErrUniqueNvNameRequired** exception will be thrown. Note that OpenLNS does not check for duplicate network variable user names when you use the *MoveToInterface* method to move a *LonMarkObject* or *NetworkVariable* to a custom interface. You would then only be able to retrieve the first network variable using the duplicated name from interface. As a result, make sure all network variables on your network have

	unique user names. If you use this method to add a network variable to a collection on a static *LonMarkObject* or a device's main interface, then the **LCA#119 lcaErrInterfaceNotModifyable** exception will be thrown, unless the device supports the addition of dynamic network variables to static *LonMarkObject* objects. You can check if a *LonMarkObject* is static or dynamic by reading the object's *IsDynamic* property.
Availability	Local, full, and lightweight clients.
Syntax	*newNV* = *networkVariables*.**Add** *nvName, nvType, nvDirection, nvOptions, memberNumber, manufacturerAssigned* **Element** — **Description** *newNV* — The *NetworkVariable* object returned by the function. This is the newly defined network variable. *networkVariables* — The *NetworkVariables* collection object being acted upon. *nvName* — The name of the network variable you are adding. The name can be a maximum of 16 characters, and it must not start with a number. In addition, it cannot include square brackets ([]), commas (,), or periods (.). *nvType* — A *TypeSpec* object identifying the type of the network variable being added. *nvDirection* — The direction of the network variable being added (input or output). The valid values for this element, which are contained in the *ConstNvDirections* constant, are as follows: **0 lcaNvDirectionInput** This value indicates that a network variable is an input network variable. **1 lcaNvDirectionOutput** This value indicates that a network variable is an output network variable. *nvOptions* — The options to use when adding the network variable to the collection. These options determine the default messaging service to be

		used for updates to the network variable, as well as the network variable's default polling, synchronization, priority and authentication settings. The valid values for this element, which are contained in the *ConstNvOptionsFlags* constant, are as follows (these options can be ORed together): **Note**: Values **0**,**1** and **2** in this constant are mutually exclusive. Selecting more than one of them will cause the **LCA, #152 lcaErrInvalidServiceType** exception to be thrown. **0 lcaNvOptionsServiceAckd** Select this option to use the acknowledged messaging service as the default messaging service for updates to the network variable. When the network variable is part of a connection, the value of the *ServiceType* property in the connection's *ConnectDescTemplate* may override this setting. **1 lcaNvOptionsServiceUnackdRpt** Select this option to use the unacknowledged, repeat messaging service as the default messaging service for updates to the network variable. When the network variable is part of a connection, the value of the *ServiceType* property in the connection's *ConnectDescTemplate* may override this setting. **2 lcaNvOptionsUnackd** Select this option to use the unacknowledged messaging service as the default messaging service for updates to the network variable. When the network variable is part of a connection, the value of the *ServiceType* property in the connection's *ConnectDescTemplate* may override this setting.

		4 lcaNvOptionsPolled Select this option to enable the network variable's polling attribute. **8 lcaNvOptionsSynchronized** Select this option to create the network variable as a synchronous network variable, meaning that all all updates to the network variable will be delivered to the application, not just the most recently received update (when earlier updates have been queued, but have not yet not delivered to the application). **16 lcaNvOptionsPriority** Select this option to use priority messaging when sending updates and polling messages to this network variable by default. When the network variable is part of a connection, the value of the *UsePriorityFlag* property in the connection's *ConnectDescTemplate* may override this setting. **32 lcaNvOptionsAuthentication** Select this option to use authenticated messaging when sending update and polling messages for this network variable by default. When the network variable is part of a connection, the value of the *UseAuthenticationFlag* property in the connection's *ConnectDescTemplate* may override this setting.
	memberNumber	A Long value indicating the member number that will be used by the network variable. The member number must be a unique value in the range of 1–4096.
	manufacturerAssigned	A Boolean value indicating whether the member number assigned to the network variable is a manufacturer-assigned member number (TRUE), or a LonMark-assigned member number (FALSE). This parameter only applies if the *NetworkVariables* collection being

	acted upon was obtained from a *LonMarkObject*; otherwise, it is ignored.
Added to API	LNS Release 3.20.

ItemByIndex

Summary	Retrieves a *NetworkVariable* object from a *NetworkVariables* collection. The object to be retrieved must be specified by its index value. This property is only supported by *NetworkVariables* collections that belong to *Interface* or *LonMarkObject* objects. If this method is invoked on a different type of network variable collection (one accessed through the *NVHubs* or *NVTargets* property), an invalid type exception (LCA #2) is raised. If this exception is encountered, it indicates that the specified collection does not support the *ItemByIndex* method. You can use this method when working with a dynamic network variable, which has an OLE collection index and name that may change because of the addition or removal of other dynamic network variables within its collection.
Availability	Local, full, and lightweight clients.
Syntax	*itemObject = itemsColl.ItemByIndex index* **Element** — **Description** *itemObject* — The *NetworkVariable* object retrieved from the collection. *itemsColl* — The *NetworkVariables* collection to be acted on. *index* — An Integer value specifying the *Index* property of the *NetworkVariable* object to be retrieved.
Added to API	LNS Release 3.0.

ItemByProgrammaticName

Summary	Retrieves a *NetworkVariable* object from a *NetworkVariables* collection by its programmatic name. You can determine the programmatic name of a *NetworkVariable* object by reading its *ProgrammaticName* property. Alternatively, you can retrieve a *NetworkVariable* object from this collection by their user names using the *Item* property.
Availability	Local, full, and lightweight clients.

Syntax	*retrievedObject = collection.ItemByProgrammaticName progName* **Element** **Description** *retrievedObject* The *NetworkVariable* object to be retrieved from the collection. *collection* The *NetworkVariables* collection being acted upon. *progName* The *ProgrammaticName* of the *NetworkVariable* object to be retrieved.
Added to API	LNS Release 3.20.

Remove

Summary	Removes a *NetworkVariable* object from the *NetworkVariables* collection. You must specify the network variable to be removed by its name, or by its index number within the collection. You can use this method to remove network variables from collections that are contained by dynamic *LonMarkObject* objects, or by custom *Interface* objects. The *indexName* element is a Variant type that allows you to specify the network variable to be deleted by its user name or by the index value assigned to it within the *NetworkVariables* collection. **Note**: The index value assigned within the *NetworkVariables* collection is not the same as the *Index* property, which stores the device index value of the network. You can determine the user name of a network variable by reading its *Name* property. If you use the user name to identify the network variable, it is important to make certain that you do not specify a user name that is shared by more than one network variable. You should use the *RemoveByIndex* method to remove a network variable. This method allows you to specify the *NetworkVariable* to be removed by its device index number, which is guaranteed to be a static, unique value. If the network variable is a member of a *LonMarkObject* , it will be unassigned from the *LonMarkObject* as it is deleted. If the network variable is part of a connection, it will not be deleted, and the **NS, #164 lcaErrNsNvmtInUse** exception will be thrown.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*networkVariables.***Remove** *indexName* **Element** **Description**

	networkVariables	The *NetworkVariables* collection to be acted upon.
	indexName	A Long value specifying the collection index of the *NetworkVariable* object to remove, or a String value specifying the name of the *NetworkVariable* object to remove.
Added to API	LNS Release 3.20.	

RemoveByIndex

Summary	Removes a *NetworkVariable* from the collection. You can specify the network variable to be deleted by its index number within the collection. You can use this method to remove network variables from collections that are contained by dynamic *LonMarkObject* objects, or by custom *Interface* objects. You can determine the device index number assigned to a network variable by reading its *Index* property. If the network variable being removed is a member of a *LonMarkObject*, it will be unassigned from the *LonMarkObject* as it is removed from the collection. You can assign network variables to *LonMarkObject* objects using the *AssignNetworkVariable* method. If the network variable is part of a connection, it will not be deleted, and the **NS, #164 lcaErrNsNvmtInUse** exception will be thrown.	
Availability	Local, full, and lightweight clients.	
Syntax	*networkVariables*.**RemoveByIndex** *deviceIndex*	
	Element	**Description**
	networkVariables	The *NetworkVariables* collection to be acted upon.
	deviceIndex	The device index number assigned to the network variable being removed.
Added to API	LNS Release 3.20.	

Properties

The *NetworkVariables* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *NetworkVariables* object in the *ConstClassIds* constant: **23 lcaClassIdNetworkVariables** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *NetworkVariable* object from a *NetworkVariables* collection. You can retrieve a *NetworkVariable* object from its

	NetworkVariables collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *NetworkVariable* object in *NetworkVariables* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *NetworkVariable* object retrieved from the collection. *collObject* — The *NetworkVariables* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *NetworkVariable* object to be retrieved. *stringExpression* — A string type specifying the name of the *NetworkVariable* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.

Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

NvMonitorOptions

The *NvMonitorOptions* object contains a set of monitoring options that can be applied to the network variable monitor points that LNS is monitoring. Depending on how this object is accessed, writing to it may affect the monitoring options used for an entire monitor set, or for an individual network variable monitor point.

The *NvMonitorOptions* object accessed through a *MonitorSet* object's *NvOptions* property contains the defaults options for the monitor set. These are the default options applied to all *NvMonitorPoint* objects as they are added to the monitor set.

You can change the defaults an individual *NvMonitorPoint* uses by writing to the *NvMonitorOptions* object accessed through the *DefaultOptions* property of that *NvMonitorPoint* object. This object contains the default options that will be used each time the monitor set containing that particular *NvMonitorPoint* object is opened. Note that this only applies to network variable monitor points in permanent monitor sets. The

CurrentOptions properties of network variable monitor points in temporary monitor sets are not accessible, as temporary monitor sets are only used in a singe client session.

You can change the monitoring options a *NvMonitorPoint* will use for an active monitoring session by writing to the *NvMonitorOptions* object accessed through the *NvMonitorPoint* object's *CurrentOptions* property. Changes made to the current options take effect for the current session only, and are not stored persistently in memory.

Note that the *NvMonitorOptions* object contained within the *NvOptions* property of a monitor set (or the *DefaultOptions* or *CurrentOptions* properties of a monitor point) is not passed by reference. If you acquire an *NvMonitorOptions* object through any of these properties and modify it, you must then explicitly assign the modified object back to the source property for the changes to take effect.

The following table summarizes the *NvMonitorOptions* object.

Description	A set of monitoring options that can be applied to the network variable points being monitored by OpenLNS.
Added to API	LNS Release 3.0.
Accessed Through	*MonitorSet* object. *NvMonitorPoint* object.
Default Property	None.
Methods	None.
Properties	• *Authentication* • *ClassId* • *GenerateInitialFetch* • *Parent* • *PollInterval* • *Priority* • *ReportByException* • *ResetPollingIfUpdated* • *Retries* • *ServiceType* • *SuppressPollingIfBound* • *ThrottleInterval* • *UseAsyncSend* • *UseBoundUpdates*

Methods

The *NvMonitorOptions* object does not contain any methods.

Properties

The *NvMonitorOptions* object contains the following properties:

- *Authentication*
- *ClassId*
- *GenerateInitialFetch*
- *Parent*
- *PollInterval*
- *Priority*
- *ReportByException*

- *ResetPollingIfUpdated*
- *Retries*
- *ServiceType*
- *SuppressPollingIfBound*
- *ThrottleInterval*
- *UseAsyncSend*
- *UseBoundUpdates*

Authentication

Summary	Determines whether the authenticated service will be used when sending a network variable monitor point updates using the *Value* property. See the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide* for more information on network variable monitor points.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*authValue* = *monOptsObject*.**Authentication** **Element** — **Description** *authValue* — Boolean value. *monOptsObject* — The *NvMonitorOptions* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *MsgMonitorOptions* object in the *ConstClassIds* constant: **79 lcaClassIdNvMonitorOptions** *object* — The object to be acted on.

Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

GenerateInitialFetch

Summary	Determines if the network variable value is fetched automatically when the network variable monitor point is enabled.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*initialFetchValue* = *nvMonOptObject*.**GenerateInitialFetch** **Element** — **Description** *initialFetchValue* — An Integer value indicating whether the network variable value is fetched automatically when the network variable monitor point is enabled. **1** The *NvMonitorPoint* object automatically gets the value of the network variable it monitors when it is enabled using the *Enable* method, implicitly enabled when it is opened, and the *doEnable* element is set to True. **0** The value will not be updated until an update is received, either due to normal polling activity or via bound updates. This is the default. **-1** The value is set to the value currently stored in the *NvOptions* property of the *MonitorSet* object. This may be useful if you want the value of the property used by a network variable monitor point to always match the default value stored in the *MonitorSet* object. This feature, however, is not available in server-independent mode. *nvMonOptObject* — The *NvMonitorOptions* object to be acted on.
Data Type	Integer.

Read/Write	Read/write.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PollInterval

Summary	Determines the length between polls for polled network variable monitoring. When polling is enabled for a monitor point, OpenLNS will periodically read the value of the monitor point and report the value using the *OnNvMonitorPointUpdateEvent*. This is most efficient when the value must be checked regularly, but the application does not need to know immediately if the value changes (for example, outside air temperature). You can enable polling for a monitor set or monitor point with the *Enable* method. • If this property is contained in the *NvMonitorOptions* object accessed through a *MonitorSet* object, this property sets the default poll interval for al monitor points contained in the set. • If this property is contained in the *NvMonitorOptions* object accessed through the *DefaultOptions* or *CurrentOptions* property of a *NvMonitorPoint* object, this property sets the default or current poll interval for that monitor point. Set the value to **lcaDefaultMcpInterval(-1)** (from the *ConstLNSMonitorSetIntervalDefault* constant) to have the poll interval default to the value contained in the

	NvOptions property of the *MonitorSet* object. This may be useful if you want the poll interval for the monitor point to always match the default value stored in the *MonitorSet* object. However, this feature is not available in server-independent mode. When writing to this property, the amount of polling you can do on a network depends on the speed of the communication channels being used by the network, and the amount of traffic on those channels. The poll interval you select combined with the number of network variables being polled by the OpenLNS Server may significantly increase the amount of traffic on your network. When you write to this property, ensure that the poll interval you specify does not cause your network resources to be exceeded.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*pollIntValue* = *nvMonOptsObject*.**PollInterval** **Element** — **Description** *pollIntValue* — The poll interval in milliseconds. *nvMonOptsObject* — The *NvMonitorOptions* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Priority

Summary	Determines whether priority is used when polling or explicitly updating a network variable monitor point using the *Value* property. Setting this property through the *NvOptions* property of a permananent monitor set, or through the *DefaultOptions* property of a permanent network variable monitor point, does not have an effect. The default priority setting for each permanent network variable monitor point is established by the *UsePriorityFlag* property of the *ConnectDescTemplate* specified when the network variable monitor point was created. You cannot set this property through the *DefaultOptions* property of a temporary network variable monitor point, as the *DefaultOptions* properties of all temporary monitor points are not accessible. However, you can set this property through the *NvOptions* property of a temporary monitor set to determine the default priority setting that will be applied to all network variable monitor points as they are added to the set.

	You can set this property through the *CurrentOptions* property of a permanent or temporary network variable monitor point to determine the priority setting to use when sending network variable updates during the current monitoring session. See the *Monitor and Control* section of the *OpenLNS Programmer's Guide* for more information.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*priorityValue* = *monOptsObject*.*Priority* **Element** — **Description** *priorityValue* — A Boolean value. **TRUE**. Priority is used when polling or explicitly updating a message monitor point using the *Value* property. **FALSE**. Priority is not used. *monOptsObject* — The *NvMonitorOptions* to be acted on.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ReportByException

Summary	Determines whether update events will only be reported when the value changes.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*byExceptionValue* = *nvMonOptsObject*.**ReportByException** **Element** — **Description** *byExceptionValue* — An Integer value indicating whether update events will only be reported when the value changes. **1** OpenLNS will only report update events to the client if the data has changed from what was last reported. **0** The value will not be updated until an update is received, either due to normal polling activity or via bound updates.

	This is the default. **-1** The value is set to the value currently stored in the *NvOptions* property of the *MonitorSet* object. This occurs if this property is contained in the *NvMonitorOptions* object accessed through the *DefaultOptions* or *CurrentOptions* property of a *NvMonitorPoint* object. This value is stored in the *ConstLNSMonitorSetOptionDefault* constant. This may be useful if you want the value of the property used by a network variable monitor point to always match the default value stored in the *MonitorSet* object. This feature, however, is not available in server-independent mode. *nvMonOptsObject* The *NvMonitorOptions* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ResetPollingIfUpdated

Summary	Determines whether the poll interval will be reset whenever a new value for the monitor point using this options set is received, either via a bound update or a read operation. The *SuppressPollingIfBound* property turns off polling if there is a bound monitoring connection.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*resetPollValue* = *nvMonOptsObject*.**ResetPollingIfUpdated** **Element** **Description** *resetPollValue* An Integer value indicating whether the poll interval is reset when a new value for the monitor point using this options set is received. **1** The polling interval will be reset if an update for the monitor point is received through a bound

	connection, or via a read operation. **0** The polling of the monitor point will continue at the prescribed interval, even if the value of the point is received between polling periods. This is the default. **-1** The value is set to the value currently stored in the *NvOptions* property of the *MonitorSet* object. This occurs if this property is contained in the *NvMonitorOptions* object accessed through the *DefaultOptions* or *CurrentOptions* property of a *NvMonitorPoint* object. This value is stored in the *ConstLNSMonitorSetOptionDefault* constant. This may be useful if you want the value of the property used by a network variable monitor point to always match the default value stored in the *MonitorSet* object. This feature, however, is not available in server-independent mode. *nvMonOptsObject* The *NvMonitorOptions* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Retries

Summary	Specifies the number of retries to use for acknowledged, request/response, or repeated service messages. This applies to network management messages only. Setting this property through the *NvOptions* properties of a permananent monitor set, or through the *DefaultOptions* property of a permanent monitor point, does not have an effect. The default retry count to use for all permanent network variable and message monitor points is established by the *RetryCount* property of the *ConnectDescTemplate* specified when the monitor point was created. You cannot set this property through the *DefaultOptions* property of a temporary monitor point because the

	DefaultOptions properties of all temporary monitor points are not accessible. However, you can set this property through the *NvOptions* property of a temporary monitor set to determine the default retry count that will be applied to all temporary monitor points as they are added to the set. You can also set this property through the *CurrentOptions* property of a permanent or temporary monitor point to determine what retry count for the current monitoring session. See the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide* for more information
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*retryValue* = *monOptsObject*.**Retries** **Element** — **Description** *retryValue* — The number of retries. The valid range of values for this property is 1–15. This property applies to network management messages only. *monOptsObject* — The *NvMonitorOptions* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

ServiceType

Summary	Determines the service type used when sending explicit messages with a message monitor point. You cannot set this property through the *CurrentOptions* property of a temporary monitor point, as the *CurrentOptions* properties of all temporary monitor points are not accessible. However, you can set this property through the *NvOptions* property of a temporary monitor set to determine the default messaging service that will be applied to all monitor points as they are added to the set. See the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide* for more information.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*retryValue* = *monOptsObject*.**Retries** **Element** — **Description**

	retryValue	The valid values for this property, which are enumerated in the *ConstServiceTypes* constant, are as follows: **0 lcaSvcAckd** Acknowledged messaging service. The device sends an acknowledgment message after it has received the message. If your application will be sending messages to large numbers of devices at once, one of the unacknowledged messaging services may be desirable, as the acknowledgment messages may generate a significant amount of network traffic. **1 lcaSvcUnackdRpt** Unacknowledged repeat messaging service. The device does not send acknowledgment messages; however, repeat messages are sent to the device after the initial message is sent to it to ensure that it reaches its destination. You can set the number of repeat messages to send, and the interval at which they will be sent, by writing to the *RepeatCount* and *RepeatTimer* properties. **2 lcaSvcUnackd** Unacknowledged messaging service. The device does not send acknowledgment messages. Do not use this service type on channels that support altnerate frequencies because the message will only be sent using the primary path. See the *AltPathType* property for more information. **3 lcaSvcRequest** Request/Response messaging service. You can use this value when sending explicit messages if the device receiving the message is designed to send a response message for the specified message code.

	monOptsObject The *NvMonitorOptions* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

SuppressPollingIfBound

Summary	Determines whether polling will be turned of if the network variable is bound to the host.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*suppressValue* = *nvMonOptsObject*.**SuppressPollingIfBound** **Element** — **Description** *suppressValue* — The *MsgMonitorOptions* object being acted upon. *asyncFlag* — An Integer value indicating whether polling will be turned of if the network variable is bound to the host. **1** Polling will be suppressed if the network variable is bound to the host. **0** Polling will not be suppressed. **-1** The value is set to the value currently stored in the *NvOptions* property of the *MonitorSet* object. This occurs if this property is contained in the *NvMonitorOptions* object accessed through the *DefaultOptions* or *CurrentOptions* property of a *NvMonitorPoint* object. This value is stored in the *ConstLNSMonitorSetOptionDefault* constant. This may be useful if you want the value of the property used by a network variable monitor point to always match the default value stored in the *MonitorSet* object. This feature, however, is not available in server-independent mode.
Data Type	Boolean.

Read/Write	Read/write. **Note**: OpenLNS sets this property automatically for message monitor points. For example, if you are sending a message via the *SendMsgWait* method, a response from the device is expected. Therefore, OpenLNS will set the *UseAsyncSend* property to False. Also, if you are writing to a message monitor point via the *OutputDataPoint* property, no response is expected; therefore, OpenLNS sets the property to True. As a result, you should not write to this property when you access it through a *MsgMonitorOptions* object.
Added to API	LNS Release 3.0.

ThrottleInterval

Summary	Determines the throttle for network variable updates. The throttle is the minimum interval between updates to the client. This is independent of the poll interval (see the *PollInterval* property).
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*throttleValue* = *nvMonOptsObject*.**ThrottleInterval** **Element** — **Description** *throttleValue* — The throttle value in milliseconds. If this property is contained in the *NvMonitorOptions* object accessed through the *DefaultOptions* or *CurrentOptions* property of a *NvMonitorPoint* object, you can set the value to **lcaDefaultMcpInterval (-1)** (from the *ConstLNSMonitorSetIntervalDefault* constant) to have the throttle interval default to the value contained in the *NvOptions* property of the *MonitorSet* object. This may be useful if you want the throttle interval for the monitor point to always match the default value stored in the *MonitorSet* object. However, this feature is not available in server-independent mode. *nvMonOptsObject* — The *NvMonitorOptions* object to be acted on.
Data Type	Long.
Read/Write	Read/write.

Added to API	LNS Release 3.0.

UseAsyncSend

Summary	Determines whether OpenLNS will wait for a completion code to return after updating the value of a monitor point before sending its next update message. When sending the values of network variable and message monitor points prior to LNS 3.20, LNS would wait for the completion code for each message sent to be returned before returning to the user, regardless of the messaging service type being used. You can use this property to determine whether OpenLNS should wait for the completion code. **Note:** The *UseAsyncSend* property can be used to confirm when the values of the monitor points on your network have been successfully sent, as described earlier. However, you can also use the *OnMsgMonitorPointErrorEvent* and *OnNvMonitorPointErrorEvent* events to determine when values are not successfully sent. These events are generated whenever there is a write failure on a monitor point. See the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide* for more information.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*monOptsObject*.**UseAsyncSend =** *asyncFlag* **Element** — **Description** *monOptsObject* — The *NvMonitorOptions* object being acted upon. *asyncFlag* — A Boolean value indicating whether OpenLNS waits for a completion code to return after updating the value of the monitor point before sending its next update message. **TRUE**. OpenLNS does not wait for the completion code to return after sending the values of the monitor points before returning to the user. In this case, OpenLNS will generate an *OnMsgMonitorPointEvent* or *OnNvMonitorPointEvent* event as soon as the completion code has been returned, and the value of the monitor point has been updated. You can use these

	events to confirm that the values of your monitor points have been successfully sent. This approach may be useful if you are updating a large number of monitor points at once, and do not want to wait for a completion code to return after each update before moving to the next one. Setting this property to True therefore may be useful when writing to the values of large numbers of data points with the *Write* method because it reduces the time required to update the values on the network. **FALSE**. OpenLNS waits for the completion code to return after it sends each value before returning from the method, as with LNS versions prior to LNS 3.20. Other client applications will be able to successfully update the monitor point while your application is waiting for the completion code. If this property is set to False and no completion code is returned after a message is sent, this indicates that LNS failed to update the monitor point. In this case, the **DS, #411 lcaErrLnsDsWriteFailed** exception will be thrown before LNS proceeds to the next update message. OpenLNS will not return any events confirming that the value has been updated when the property is set to False. This is the default.
Data Type	Boolean.

Read/Write	Read/write. **Note**: OpenLNS sets this property automatically for message monitor points. For example, if you are sending a message via the *SendMsgWait* method, a response from the device is expected. Therefore, OpenLNS will set the *UseAsyncSend* property to False. Also, if you are writing to a message monitor point via the *SendMsgWait* property, no response is expected; therefore, OpenLNS sets the property to True. As a result, you should not write to this property when you access it through a *MsgMonitorOptions* object
Added to API	LNS Release 3.20.

UseBoundUpdates

Summary	Indicates whether *NvMonitorPoint* objects should use bound updates for monitor and control. Enabling bound monitoring and control will not disable polling unless the *SuppressPollingIfBound* is set to true. This property enables implicit binding. With implicit binding enabled, OpenLNS attempts to create connections without user intervention. Attempting to set this property in the *CurrentOptions* object will fail silently for both permanent and temporary monitor points. To use bound updates for a permanent monitor point, you must set this property in the *DefaultOptions* object. If you are in communication with the OpenLNS Server (not in server-independent mode), this option will be automatically enabled as soon as you set the property to True. In this case, you don't have to close and re-open the monitor set, as with other options in the *DefaultOptions* object. You cannot set the properties of the *DefaultOptions* object in server-independent mode. OpenLNS does not support the automatic connection of monitor points in temporary monitor sets; therefore, this property should be set to False for *NvMonitorOptions* objects that are accessed through the *NvOptions* property of *MonitorSet* objects created as temporary monitor sets.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*useBoundFlag* = *nvMonOptsObject*.**UseBoundUpdates** **Element** — **Description** *useBoundFlag* — A Boolean value indicating whether the monitor points using this set of options should use bound updates. **TRUE**. The monitor points using

		this set of options should use bound updates. **FALSE**. The monitor points using this set of options should not use bound updates.
	nvMonOptsObject	The *NvMonitorOptions* object to be acted on.
Data Type	Boolean.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

NvMonitorPoint

A *NvMonitorPoint* object represents a single monitored or controlled network variable. You can use network variable monitor points to perform explicit reading and writing, polled monitoring, or implicit bound monitoring of network variables on devices in your network with your application. For more information on network variable monitor points, see the *Monitor and Control* chapter in the *OpenLNS Programmer's Guide*.

In general, the *NvMonitorPoint* objects contained in *MonitorSet* objects created as temporary or permanent monitor sets behave the same way except that the *DefaultOptions* properties of the *NvMonitorPoint* objects in temporary monitor sets are not accessible. For more information on this, see the *DefaultOptions* property of this object.

In addition, network variable monitor points in temporary monitor sets cannot be automatically bound to the monitoring node. This means that the *UseBoundUpdates* property stored in the *NvMonitorOptions* objects used by all temporary monitor points must be set to False. For more information on this, see the online help for the *UseBoundUpdates* property.

Temporary monitor sets, and all the monitor points they contain, can only be used in a single client session. They are deleted by OpenLNS as soon as the session in which they were created ends. For more information on the differences between temporary and permanent monitor sets, see the *MonitorSet object*.

The following table summarizes the *NvMonitorPoint* object.

Description	A single monitored or controlled network variable.
Added to API	LNS Release 3.0.
Accessed Through	*NvMonitorPoints* object.
Default Property	*Name*.
Methods	• *Advise* • *Disable* • *Enable* • *Unadvise*
Properties	• *ClassId* • *CurrentOptions* • *DataPoint* • *DefaultOptions* • *FormatSpec*

	• *Name* • *Parent* • *Tag*
Events	• *UpdateErrorEvent* • *UpdateEvent*

Methods

The *NvMonitorPoint* contains the following methods.

- *Advise*
- *Disable*
- *Enable*
- *Unadvise*

Advise

Summary	Enables update and error events for an object that implements the *ILcaMsgMonitorPointListener* or *ILcaNvMonitorPointListener* interface. This method should only be used if the development environment supports multi-threading (such as Visual C++). When you call this method, OpenLNS will provide event notification of updates and update errors using callbacks, instead of Windows messaging. The callback is made to the *UpdateEvent* or *UpdateErrorEvent* method of the object specified as the *object* parameter when the *Advise* method is called. When you call the *Advise* method, the client thread will stop generating *OnNvMonitorPointErrorEvent* and *OnNvMonitorPointUpdateEvent* events for the *NvMonitorPoint* object specified as the *mpObject* element. The object specified as the *object* element will then start receiving *UpdateErrorEvent* events and message *UpdateEvent* events for that monitor point. **Note**: The *Advise* method must be called from the event handler that is managing the *NvMonitorPoint* events listed above. You should determine how these updates are handled by defining the *UpdateErrorEvent* and the *UpdateEvent* for the *ILcaNvMonitorPointListener* interface. The returned tag should be supplied to the *Unadvise* method to return update notification to the client thread.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*tag* = *mpObject*.**Advise** *object* **Element** **Description**

	tag	An Integer type
	mpObject	The *NvMonitorPoint* object to be acted on.
	object	An object that implements the *ILcaNvMonitorPointListener* interface.
Added to API	LNS Release 3.0.	

Disable

Summary	Disables monitoring of a network variable monitor point. When you disable monitoring of a network variable monitor point, you should note that this overrides subsequent calls to the *MonitorSet* object's *Enable* method. For example, if you call the *Disable* method on a *NvMonitorPoint* object named Point A, and then call the *Disable* method on the monitor set containing Point A, Point A would not be enabled. All other monitor points in the monitor set, however, would be enabled. Once you have explicitly disabled an *NvMonitorPoint* with the *Disable* method, you can only re-enable that monitor point by calling the *Enable* method on it, or by closing and re-opening the monitor set it belongs to. You can also disable an entire monitor set by calling the *Disable* method on the *MonitorSet* object. When you do this, polled and bound monitoring for all monitor points on the monitor set will be disabled. After this, none of the monitor points in the set can be enabled for monitoring until the *Disable* method has been called on the *MonitorSet* object again. For more details on opening and enabling monior sets and monitor points, see Chapter 9 of the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*object.Disable* **Element** — **Description** *object* — The *NvMonitorPoint* object to be disabled.
Added to API	LNS Release 3.20.

Enable

Summary	Enables monitoring of a network variable monitor point. You can enable monitoring of all the monitor points in a permanent or temporary monitor set at once by setting the

	doEnable element to True when you *open* the monitor set. If the *doEnable* element is set to False, you can also enable the entire monitor set later by calling the *Enable* method on the *MonitorSet* object. You can disable monitoring of an individual network variable monitor point by calling the *Disable* method on it. For more details on opening and enabling monior sets and monitor points, see Chapter 9 of the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*object*.**Ensable** **Element** — **Description** *object* — The *NvMonitorPoint* object to be enabled.
Added to API	LNS Release 3.20.

Unadvise

Summary	Returns event generation to the client thread if it was changed to another thread using the *Advise* method.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*mpObject.Unadvise tag* **Element** — **Description** *tag* — An Integer type. This *tag* parameter should use the tag that was returned when the *Advise* method was called. *mpObject* — The *NvMonitorPoint* object to be acted on.
Added to API	LNS Release 3.0.

Properties

The *NvMonitorPoint* object contains the following properties:

- *ClassId*
- *CurrentOptions*
- *DataPoint*
- *DefaultOptions*
- *FormatSpec*
- *Name*
- *Parent*
- *Tag*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *NvMonitorPoint* object in the *ConstClassIds* constant: **80 lcaClassIdNvMonitorPoint** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CurrentOptions

Summary	Contains the current monitoring options for this network variable monitor point. Once the monitor set containing a monitor point has been opened, you can use this property to set the monitoring options that will be used for the monitor point during that particular session. For monitor points in *MonitorSet* objects created as permanent monitor sets, the options contained in this property default to the options set in the *DefaultOptions* property of the *NvMonitorPoint*. If the options are not set there, they default to the options set in the *NvOptions* properties in the *MonitorSet* object. Each time a permanent *MonitorSet* object's *Open* method is called, the current options for each of the monitor points in the set are reset to the options contained in their *CurrentOptions* properties. The *CurrentOptions* property can only be written to when the monitor set is open. The *NvMonitorOptions* object contained within this property is not passed by reference. If you acquire an *NvMonitorOptions* object through the *CurrentOptions* property and modify it, you must then explicitly assign the modified object back to the *CurrentOptions* property for the changes to take effect. This following code sample

	demonstrates this procedure:
	`Set curOptions = monPoint.CurrentOptions`
	`curOptions.Authentication = True`
	`curOptions.Retries = 5`
	`Set monPoint.CurrentOptions = curOptions`

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*curOptions = monPoint.CurrentOptions* **Element** — **Description** *curOptions* — The *NvMonitorOptions* object containing the current options for this monitor point. *monPoint* — The *NvMonitorPoint* object to be acted on.
Data Type	*NvMonitorOptions* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

DataPoint

Summary	Contains a *DataPoint* object that can be used to read or write to the network variable monitor point. The monitor set containing the *NvMonitorPoint* must be open in order for you to access this property. The *DataPoint* object's *FormatSpec* property will be initialized to contain the same *FormatSpec* object as the *NvMonitorPoint* object's *FormatSpec* property. **Note**: This property does not persistently contain the same *DataPoint* object. Each time you access this property, a new *DataPoint* object will be created
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*dpObject = mpObject.DataPoint* **Element** — **Description** *dpObject* — A *DataPoint* object which provides access to the monitor point value. *systemObject* — The *NvMonitorPoint* object to be acted on.
Data Type	*DataPoint* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

DefaultOptions

Summary	Contains the default monitoring options that are applied each time the monitor set containing this network variable or network variable monitor point is opened.
	
	These options are read when the monitor set containing the monitor point is opened. Changes to these options will not take effect until the next time the monitor set is opened. Use the *CurrentOptions* property to change the active monitoring options to use for a monitor point that is currently enabled.
	
	For network variable monitor points, the options contained in the *DefaultOptions* property default to the options set in the permanent monitor set's *NvOptions* property.
	
	The default options cannot be accessed in server-independent mode; therefore, Independent clients cannot read or write to the *DefaultOptions* property.
	
	The *NvMonitorOptions* object contained within this property is not passed by reference. If you acquire an *NvMonitorOptions* object through the *DefaultOptions* property and modify it, you must then explicitly assign the modified object back to the *DefaultOptions* property for the changes to take effect. This following code sample demonstrates this procedure:
	
	`Set defOptions = monPoint.DefaultOptions`
	`defOptions.Authentication = True`
	`Set monPoint.DefaultOptions = defOptions`

Availability	Local, full, and lightweight clients.
Syntax	*defOptions = monPoint. DefaultOptions* **Element** — **Description** *defOptions* — The *NvMonitorOptions* object containing the default options for this monitor point. *monPoint* — The *NvMonitorPoint* object to be acted on.
Data Type	*NvMonitorOptions* object.
Read/Write	Read/write. **Note:** The *CurrentOptions* properties of monitor points in *MonitorSet* objects created as temporary monitor sets are not accessible. If you attempt to acquire the *CurrentOptions* property through a temporary monitor point, the **LCA, #161 lcaErrNotAllowedOnTemporaryObject** exception is thrown. This is because these monitor points can only be used in a singe client session. If you want a temporary monitor point to use options other than the defaults for the monitor set, you can change them with the *CurrentOptions* property.

Added to API	LNS Release 3.0.

FormatSpec

Summary	Contains the format specification information for data read from the *FormattedValue* property in the *DataPoint* or *NvMonitorPoint* object.
	
	This property contains a *FormatSpec* object that determines the base type to use when reading the formatted values of the data point or monitor point. For data points acquired through *NetworkVariable* objects, the default settings for the *FormatSpec* object are determined based on the network variable's *TypeSpec* property.
	
	For data points, you can also use the *CurrentFormatLocale* property to determine which options OpenLNS will use when displaying the data stored in the *DsIsDefaultFormat* property.
	
	The *FormatSpec* object contained within this property is not passed by reference. If you modify the values assigned to the properties of a local *FormatSpec* object, you must then explicitly assign the modified *FormatSpec* object back to the *FormatSpec* property of the *NvMonitorPoint* or *DataPoint* for the changes to take effect. This following code sample demonstrates this procedure:
	
	`Set fsObject = dpObject.FormatSpec`
	`fsObject.FormatName = "SNVT_temp_f#SI"`
	`Set dpObject.FormatSpec = fsObject`

Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects and temporary monitor points are not available on Independent clients.
Syntax	*fsObject= dpObject.FormatSpec* **Element** — **Description** *fsObject* — The *FormatSpec* object containing the format information. *dpObject* — The *NvMonitorPoint* object to be acted on.
Data Type	*FormatSpec* object.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period

	(.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Tag

Summary	Stores any extra data associated with the monitor point or monitor set. The data stored in this property is not used by OpenLNS, and is available as soon as the data point or monitor set is created in a monitor and control session. You can use this property to store any data your application may need when using the monitor set or monitor point. For example, you could store the name of the monitor set that a message or network variable monitor point belongs to, or the name of the

<table>
<tr><td></td><td>application device that a monitor set belongs to.

The Tag properties for all NvMonitorPoint objects in permanent monitor sets are cached when the monitor set is opened. As a result, any changes made the Tag properties of these monitor points while the permanent monitor set is open will not be accessible until the monitor set is closed and re-opened. When initially created, monitor points and monitor sets will have a null Tag value. However, if you add a monitor point to an open monitor set and set its Tag value in the same transaction, you will be able to access the Tag value during that monitor and control session, as all data would be written to the device as soon as the transaction is committed. You should note that this behavior does not apply to monitor points in temporary monitor sets. Temporary monitor sets support "live" updates to the value of the Tag property.

A well-defined monitoring application will include any information necessary to quickly and efficiently identify the monitor point in this property. This will eliminate the need to gather such information from the database, or to perform other time-consuming activities, during the monitoring process.</td></tr>
<tr><td>Availability</td><td>Local, full, lightweight, and independent clients. Note that temporary MonitorSet objects are not available on Independent clients.</td></tr>
<tr><td>Syntax</td><td>tagValue = Object.Tag

Element — Description
Object — The monitor point or monitor set object to be acted on.
tagValue — The tag associated with the object.</td></tr>
<tr><td>Data Type</td><td>Variant.</td></tr>
<tr><td>Read/Write</td><td>Read/write.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.0.</td></tr>
</table>

Events

The *NvMonitorPoint* object contains the following events:

- *UpdateErrorEvent*
- *UpdateEvent*

UpdateErrorEvent

Summary	Indicates that a write failure or a poll failure has occurred on the network variable monitor point that generated the event. This event can only be used in development environments that support events being generated on threads other than the client thread (such as Visual C++). In these environments, it is usually more efficient to generate events from a non-client thread. To cause events to be generated in this manner, create an object which implements the *ILcaNvMonitorPointListener* interface and calls an *NvMonitorPoint* object's *Advise* method with the created object as the argument. The object will now receive these events directly. The behavior of the object depends on how the user implements the *UpdateEvent* method.
Syntax	*UpdateErrorEvent*(*UpdateType* as Integer) **Element** — **Description** *updateType* — This element always returns the value 1.
Data Type	Integer.
Added to API	LNS Release 3.0.

UpdateEvent

Summary	Indicates that a network variable monitor point update has arrived. This event can only be used in development environments that support events being generated on threads other than the client thread (such as Visual C++). In these environments, it is usually more efficient to generate events from a non-client thread. To cause events to be generated in this manner, create an object which implements the *ILcaNvMonitorPointListener* interface and calls an *NvMonitorPoint* object's *Advise* method with the created object as the argument. The object will now receive these events directly. The behavior of the object depends on how the user implements the *UpdateEvent* method. For completion code messages, the *InputDp*, *OutputDp*, and *Src* parameters are NULL.
Syntax	*UpdateErrorEvent*(*UpdateType* as Integer) **Element** — **Description** *UpdateType* — This element always returns the value 0. The possible values for this parameter,

		which are contained in the *ConstMonitorEventType* constant, are as follows: **0 lcaMonitorEventTypeNull** This value is not used. **1 lcaMonitorEventTypeQuit** This value is not used. **2 lcaMonitorEventTypeAdd** This value is not used. **3 lcaMonitorEventTypeRemove** This value is not used. **4 lcaMonitorEventTypeMsCreate** This value is not used. **5 lcaMonitorEventTypeMsDelete** This value is not used. **6 lcaMonitorEventTypeMsChange** This value is not used. **7 lcaMonitorEventTypeMsError** This value is not used. **8 lcaMonitorEventTypeNvCreate** A network variable monitor point has been created. **9 lcaMonitorEventTypeNvDelete** A network variable monitor point has been removed. **10 lcaMonitorEventTypeNvChange** This value is not used. **11 lcaMonitorEventTypeNvError** A network variable monitor point has returned an error. See the *OnNvMonitorPointErrorEvent*. **12 lcaMonitorEventTypeNvUpdate** A network variable monitor point has received an update. See the *OnNvMonitorPointUpdateEvent*. **13 lcaMonitorEventTypeNvComplete** A completion code has returned for the monitor point.

		14 lcaMonitorEventTypeMsgCreate A message monitor point has been created. **15 lcaMonitorEventTypeMsgDelete** A message monitor point has been removed. **16 lcaMonitorEventTypeMsgChange** This value is not used. **17 lcaMonitorEventTypeMsgError** A message monitor point has returned an error. See the *OnMsgMonitorPointErrorEvent*. **18 lcaMonitorEventTypeMsgUpdate** A message monitor point has received an update. See the *OnMsgMonitorPointUpdateEvent*. **19 lcaMonitorEventTypeMsgRequest** A message monitor point has received a request message. **20 lcaMonitorEventTypeMsgResponse** A message monitor point has received a reponse message. **21 lcaMonitorEventTypeMsgComplete** This value is not used.
	InputDp	A *DataPoint* object containing the received value. The *InputDp* parameter's *AutoRead* and *AutoWrite* properties are set to False.
	OutputDp	A *DataPoint* object that allows a response to be sent if the *UpdateType* is Request. The *OutputDp* parameter's *AutoWrite* property is set to True; therefore, it will be sent as a response when you write to this *DataPoint* object.
	Src	A *SourceAddress* object indicating the source device of the update.
Data Type	Integer.	
Added to API	LNS Release 3.0.	

NvMonitorPoints

The *NvMonitorPoints* object contains a collection of *NvMonitorPoint* objects. This collection object contains all of the *NvMonitorPoint* objects that have been added to a monitor set. *Network* variable monitor points can be used for explicit reading and writing, polled monitoring, or implicit bound monitoring of network variables on devices in your network. For more information on network variable monitor points, see the *Monitor and Control* chapter of the *OpenLNS Programmer's Guide*.

In general, the *NvMonitorPoint* collections contained in *MonitorSet* objects that were created as permanent monitor sets, or as temporary monitor sets, behave the same way. However, there are several variances in the behavior of individual network variable monitor points you should note when using temporary monitor sets. The *DefaultOptions* properties of network variable monitor points in temporary monitor sets are not accessible. For more information on this, see the online help for the *DefaultOptions* property.

In addition, network variable monitor points in temporary monitor sets cannot be automatically bound to the monitoring node. This means that the *UseBoundUpdates* property stored in the *NvMonitorOptions* objects used by all temporary monitor points must be set to False.

The following table summarizes the *NvMonitorPoints* object.

Description	A collection of *NvMonitorPoint* objects.
Added to API	LNS Release 3.0.
Accessed Through	*MonitorSet* object.
Default Property	*Item*
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *NvMonitorPoints* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	Adds a new *NvMonitorPoint* object to the collection. You should use transactions when creating large numbers of network variable monitor points in permanent monitor sets. This is because it will reduce the time required to create the monitor points. For more information on using transactions with OpenLNS, see the *Programming an OpenLNS Application* chapter of the *OpenLNS Programmer's Guide*. Monitor points in permanent monitor sets are not

<table>
<tr><td></td><td>automatically removed when the application shuts down. If a monitor point is not going to be used again, remove it with the Remove method before closing your application. This does not apply to monitor points in temporary monitor sets, as temporary monitor sets are deleted as soon as the application that created them shuts down. As a result, you should use temporary monitor points for monitor points you will only need to use once.

Unlike temporary monitor sets, you can add network variable monitor points to a permanent monitor set while the monitor set is closed, and while your application is operating in engineered mode. Monitor points added to a permanent set are not available for monitoring until the transaction is committed, and if they are added while the MgmtMode property is set to lcaMgmtModeDeferConfigUpdates, they are not available until it is set to lcaMgmtModePropagateConfigUpdates.</td></tr>
<tr><td>Availability</td><td>Local, full, lightweight, and independent clients. Note that temporary MonitorSet objects, and temporary monitor points, are not available on Independent clients.</td></tr>
<tr><td>Syntax</td><td>nvMpObject = nvMpColl.Add nvMpName, nv_target, connDesc

Element — Description

nvMpObject — The newly defined NvMonitorPoint object.

nvMpColl — The NvMonitorPoints collection object.

nvMpName — The Name of the new NvMonitorPoint object.

nv_target — A NetworkVariable object containing the network variable to be monitored.

connDesc — A ConnectDescTemplate object that determines how network variables will be sent out through this monitor point.

You can set this parameter to NULL to use the default OpenLNS connection description template.

Note: Network variable monitor points in temporary monitor sets do not support the use of connection description templates, and so you must set this element to NULL when adding a network variable monitor point to a temporary monitor set.</td></tr>
<tr><td>Added to API</td><td>LNS Release 3.0.</td></tr>
</table>

Remove

Summary	Removes an *NvMonitorPoint* object from the collection. This method can only be called when connected to the network database (after the *Network* object's *Open* method has been called). This method only applies to network variable monitor points in permanent monitor sets. When an OpenLNS application is closed, all temporary monitor sets and temporary network variable monitor points created by that application are deleted automatically.
Availability	Local, full, lightweight, and independent clients. Note that temporary *MonitorSet* objects, and temporary monitor points, are not available on Independent clients.
Syntax	*nvMpColl*.**Remove** *indexName* **Element** — **Description** *nvMpColl* — The *NvMonitorPoints* collection object containing the *NvMonitorPoint* object to be removed. *indexName* — A Long value specifying the collection index of the *NvMonitorPoint* object to remove, or a String value specifying the name of the *NvMonitorPoint* object to remove.
Added to API	LNS Release 3.0.

Properties

The *NvMonitorPoints* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the

	NvMonitorPoints object in the *ConstClassIds* constant: **81 lcaClassIdNvMonitorPoints** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *NvMonitorPoint* object from a *NvMonitorPoints* collection. You can retrieve a *NvMonitorPoint* object from its *NvMonitorPoints* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *NvMonitorPoint* object in *NvMonitorPoints* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** **Description**

	retrievedObject	The object retrieved from the collection.
	collObject	The collection object to be acted on.
	index	A Long type specifying the ordinal index of the object to retrieve.
	stringExpression	A string type specifying the name of the object to retrieve.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.	
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.	
Syntax	*parentObject* = *object*.**Parent**	
	Element	**Description**
	parentObject	The object that is the parent of the specified *object*.
	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For**

	Each ... Next statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

ObjectServer

An *ObjectServer* object represents a single instance of the OpenLNS ActiveX Control. The default name is **LcaObjectServer1**, but this may be changed using the *Name* property. There may be only one instance per process, but multiple processes may be run, each with a different instance. As of Release 2.00, the OpenLNS ActiveX control allows apartment model threading.

The following table summarizes the *ObjectServer* object.

Description	An instance of the OpenLNS ActiveX Control.
Added to API	Prior to LNS Release 3.0.
Methods	• *AboutBox* • *AcceptIcomingSession* • *BeginIncomingSessionEvents* • *Close* • *CompactDb* • *Drag* • *EndIncomingSessionEvents* • *ExtensionByHandle* • *Move* • *Open* • *RebuildLdrfCatalog* • *SetCustomerInfo* • *SetFocus* • *SetLicenseInfo* • *SetLicenseInfoEx* • *ShowWhatsThis* • *ZOrder*
Properties	• *ActiveNetwork* • *ActiveRemoteNI*

	• *CausesValidation* • *ClassId* • *ComponentApps* • *Container* • *CurrentFormatLocale* • *DatabasePath* • *DragIcon* • *DragMode* • *Extensions* • *Flags* • *FormatLocales* • *Height* • *HelpContextID* • *Index* • *IsOpen* • *LdrfLanguages* • *LdrfCatalogPath* • *Left* • *Name* • *NetworkInterfaces* • *Networks* • *Object* • *Parent* • *RemoteFlag* • *RemoteNetworks* • *ResourceLanguageId* • *TabIndex* • *TabStop* • *Tag* • *ToolTipText* • *Top* • *Version* • *Visible* • *VNINetworks* • *WhatsThisHelpID* • *Width*
Events	• *DragDrop* • *DragOver* • *GotFocus* • *LostFocus* • *OnAttachment* • *OnChangeEvent* • *OnCommissionEvent* • *OnDbConversionEvent* • *OnDbValidationEvent* • *OnIncomingSessionEvent* • *OnLonMarkObjectStatusChangeEvent* • *OnMissedEvent* • *OnMsgMonitorPointErrorEvent* • *OnMsgMonitorPointEvent* • *OnMsgMonitorPointUpdateEvent* • *OnNetworkServiceDeviceResetNew*

	• *OnNetworkVariableStringUpdate* • *OnNetworkVariableUpdate* • *OnNodeConnChangeEvent* • *OnNodeIntfChangeEvent* • *OnNvMonitorPointErrorEvent* • *OnNvMonitorPointEvent* • *OnNvMonitorPointUpdateEvent* • *OnNVUpdateError* • *OnSessionChangeEvent* • *OnSystemMgmtModeChangeEvent* • *OnSystemNssIdle* • *OnSystemServicePin* • *Validate*

Methods

The *ObjectServer* object contains the following methods.

- *AboutBox*
- *AcceptIcomingSession*
- *BeginIncomingSessionEvents*
- *Close*
- *CompactDb*
- *Drag*
- *EndIncomingSessionEvents*
- *ExtensionByHandle*
- *Move*
- *Open*
- *RebuildLdrfCatalog*
- *SetCustomerInfo*
- *SetFocus*
- *SetLicenseInfo*
- *SetLicenseInfoEx*
- *ShowWhatsThis*
- *ZOrder*

AboutBox

Summary	Raises an About Box accessible in design mode that contains copyright information. Typically, in a development environment, a button or field can be selected on the property page to show the about box with the current information.
Availability	Local, full, lightweight, and independent clients.
Syntax	*osObject*.**AboutBox** **Element** — **Description** *osObject* — The *ObjectServer* object to be acted on.
Added to API	Prior to LNS Release 3.0.

AcceptIncomingSession

Summary	Accepts or rejects an incoming uplink session request after you have registered your application for uplink session handling. You can use this method to accept or reject an uplink session request after a request has been received, and the *OnIncomingSessionEvent* event has been fired. If an uplink session request is rejected, the session will be terminated. If the uplink session request is neither accepted nor rejected before the session establishment time-out period for the xDriver Profile handling the session expires, the request will be rejected automatically. The session establishment time for a Profile can be configured using the xDriver Profile Editor. For more information on this, see Chapter 3 of the *OpenLDV Programmer's Guide, xDriver Supplement*. Note that your application must register to receive uplink session requests by invoking the *BeginIncomingSessionEvents* method. Once you have accepted a session, you can open the network containing the RNI device involved in the session, and use the *OnSessionChangeEvent* event to monitor the state of the xDriver session. These methods and events only apply to clients that are using the OpenLDV xDriver to connect to remote network interface (RNI) devices. For an overview of the OpenLDV xDriver, see the *OpenLNS Network Interface Drivers* chapter in the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients.
Syntax	*objServer*.**AcceptIncomingSession** *tag, acceptUplink, postponeUpdates* **Element** — **Description** *osObject* — The *ObjectServer* object to be acted on. *tag* — A Long value identifying the session being accepted or rejected. This value is passed to the *OnIncomingSessionEvent* event as the *tag* element when the uplink session request is received. *acceptUplink* — A Boolean value indicating whether the session is to be accepted. **TRUE**. Accepts the incoming uplink session request. **FALSE**. Rejects the session. *postponeUpdates* — A Boolean value indicating whether monitor point update events are to be suspended while the uplink session is

	being opened. **TRUE**. Suspends all monitor point update events while the uplink session is being opened. In this case, withheld monitor point updates must be released by calling the *ReleasePendingUpdates* method. **FALSE**. Deletes all monitor point update events that occur while the session is being initialized.
Added to API	LNS Release 3.06.

BeginIncomingSessionEvents

Summary	Registers your application for incoming session event handling. Once you have invoked this method, your application will be notified of incoming uplink session requests to the OpenLNS Server via the *OnIncomingSessionEvent* event. You can invoke this method multiple times per application if you want to use multiple Profiles to listen for incoming session requests in a single application. However, multiple applications cannot register for uplink session event handling with the same Profile simultaneously. After you have registered your application for incoming session handling with this method, the *OnIncomingSessionEvent* event will be fired each time a request for connection is received. You must then accept or reject each incoming uplink session with the *AcceptIncomingSessionEvent* method. These methods and events only apply to clients that are using the OpenLDV xDriver to connect to remote network interface (RNI) devices. For an overview of the OpenLDV xDriver, see Chapter 11, *OpenLNS network interface Drivers,* of the *OpenLNS Programmer's Guide.*
Availability	Local, full, lightweight, and independent clients.

Syntax	*objServer*.**AcceptIncomingSession** *tag, acceptUplink, postponeUpdates* **Element** — **Description** *objServer* — The Object Server object being acted upon. *xDriverProfileName* — An xDriver Profile name as a String. Your application will be informed of incoming session requests that come in on the TCP listener port assigned to the selected Profile. You can use the OpenLDV xDriver Profile Editor to create an xDriver Profile, enable it for incoming session handling, and assign it a listener port. For more information on this, see Chapter 3, *Extending the Default xDriver Profile*, of the *OpenLDV Programmer's Guide, xDriver Supplement*. *acceptUplink* — A Boolean value indicating whether the session is to be accepted. **TRUE**. Accepts the incoming uplink session request. **FALSE**. Rejects the session.
Added to API	LNS Release 3.06.

Close

Summary	Closes the OpenLNS global database.
Availability	Local, full, lightweight, and independent clients.
Syntax	*objServObject.Close* **Element** — **Description** *objServObject* — The *ObjectServer* object to be closed.
Added to API	Prior to LNS Release 3.0.

CompactDb

Summary	Defragments and re-indexes OpenLNS global database. You may not call this method on an OpenLNS database that is open and in use by any client application. You should backup all OpenLNS databases before calling this method. Also, your computer should have at least twice as much free disc space as the size of the database when you call this method.
Availability	Local, full, lightweight, and independent clients.
Syntax	*objServObject*.**CompactDb** **Element** — **Description** *objServObject* — The *ObjectServer* object to be closed.
Added to API	Prior to LNS Release 3.0.

Drag

Summary	A standard ActiveX control method used for visual controls. This method is not applicable for the Object Server.

EndIncomingSessionEvents

Summary	Disables uplink session event handling for your application. You should call this method before closing an application that has registered for uplink session handling with the *BeginIncomingSessionEvents* method, or when you no longer want the application to be responsible for handling incoming sessions. When you call the *BeginIncomingSessionEvents* method to register for uplink session handling, you will select an xDriver Profile to receive the uplink session requests with. You should call the *EndIncomingSessionEvents* method for each xDriver Profile used to receive uplink session requests before closing an application. These methods only apply to clients that are using the OpenLDV xDriver to connect to remote network interface (RNI) devices. For an overview of the OpenLDV xDriver, see the *OpenLNS Network Interface Drivers* chapter of the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients.
Syntax	*objServer.EndIncomingSessionEvents xDriverProfileName* **Element** — **Description** *objServObject* — String containing the name of the xDriver Profile used in the call to *BeginIncomingSessionEvents* method.
Added to API	LNS Release 3.06.

ExtensionByHandle

Summary	Retrieves an *Extension* object by its *Handle* property. This method can be used to retrieve an extension record in the global database—regardless of the collection containing the *Extension* object. This may be particularly useful when processing events reported by *OnChangeEvent* with objectType of **lcaChangeEventExtensions**, and **networkHandle** equal to 0.
Availability	Local, full, and lightweight clients.
Syntax	*extension* = *objectServer*.**ExtensionByHandle** **Element** — **Description** *Extension* — The *Extension* object. *system* — The *ObjectServer* object. *handle* — The handle of the *Extension* object to be retrieved.
Added to API	OpenLNS.

Move

Summary	This is a standard ActiveX control property used for visual controls. This property is not applicable for the Object Server

Open

Summary	Opens the OpenLNS global object server database. To initialize the ObjectServer, an application must do the following (see the *OpenLNS Programmer's Guide* for more detailed information): 1. Open the global database using the *Open* method of the *ObjectServer*. 2. Select or create a *Network* object. 3. Open the network database using the *Open* method of the *Network* object. 4. Select or create a *System* object. 5. Initialize the *System* object by using the *Open* method of the *System* object. Invoking the *Open* method of the *ObjectServer* opens the global database. The database path name is read from the Windows System Registry which may be set using the *DatabasePath* property of the *ObjectServer*. Before invoking this method, the application needs to specify whether it is a remote client by setting the *RemoteFlag* property. In addition, you can use the *Flags* property of the ObjectServer to specify a series of global flags that determine how LNS will behave with the client application's processes, including the type of network variable update used, post-

	connection update state, remote application transfer state and the Data Server remote-IP mode used by the LNS Server. However, if you plan to modify the current value of the *Flags* property, you must do so before opening the ObjectServer with this method. If you will be *opening any networks* with an OpenLNS application that is running as a Windows service, then the first application to open the OpenLNS Object Server must also be running as a Windows service. In addition, if a network is to be opened by an OpenLNS application that is running as Windows service, then that network and system must be opened by an OpenLNS application that is running as Windows service before it is opened with an OpenLNS application running as a user process. If you open the OpenLNS Object Server or a network with a user process before opening it with a service, you must first close the network and OpenLNS Object Server before opening them with an OpenLNS application that is running as Windows service. To avoid these problems, you should automatically start services that open an OpenLNS network, and the services should open the network and systems is accessing upon startup. This ensures that the network is opened by the service before a user process.
Availability	Local, full, and lightweight clients.
Syntax	*objServObject*.**Open** **Element** **Description** *objServObject* The *ObjectServer* object to be opened.
Added to API	Prior to LNS Release 3.0.

RebuildLdrfCatalog

Summary	The LonMark Device Resource Files catalog is used to keep track of all sets of LonMark Device Resource Files that are to be referenced by OpenLNS. Prior to OpenLNS, the catalog would be found at <*LonWorks Path*>**Types** folder. With OpenLNS and the introduction of the LonWorks Data Path, the catalog is now found at <*LonWorks Data Path*>**Types**. When the catalog is refreshed using this method, any new LDRF file sets that have been added in sub-directories of either **LonWorks\Types** or <*LonWorks Data Path*>**Types** will be automatically added to the catalog. New file sets can therefore be copied in to the appropriate place in the **Types** sub-directories, and found and cataloged when this method is called. The recommended sub-directory structure for file sets is **Types\User**<*Manufacturer Name*>. Some large companies have added hierarchies under the manufacturer name as well.
Availability	Local clients only.
Syntax	*objServer*.**RebuildLdrfCatalog** *DeleteOldCatalog* **Element** — **Description** *objServer* — The *ObjectServer* object to be acted on. *DeleteOldCatalog* — A Boolean value indicating whether the current LDRF catalog should be deleted before rebuilding the contents. If this method fails to rebuild the catalog, you can recover it by setting this parameter TRUE, as the catalog may be permanently corrupted. • **TRUE**. Delete the current LDRF catalog before rebuilding. • **FALSE**. Update the current catalog.
Added to API	OpenLNS.

SetCustomerInfo

Summary	Sets the customer ID and key. By default, the OpenLNS Server allows the installation of up to four devices to a system. This does not include routers, or the NSI. To add additional devices, invoke this method with a valid customerId and customerKey. The *customerId* and *customerKey* values are printed on the back cover of the OpenLNS Standard Development Kit CD-ROM jewel case. Note that the *customerKey* parameter is case sensitive and may not contain spaces. For Local and Full client applications, you should call this

	method before opening the Object Server if you plan on running your application in the standard licensing mode (Standard Mode). For Lightweight client applications, you must always call this method prior to opening the Object Server, as Lightweight client applications can only operate in Standard Mode. Note that this is the last step a Local, Lightweight or Full client application should take before opening the Object Server. For more information on the steps you should take when initializing the Object Server, see the *OpenLNS Programmer's Guide*.
Availability	Local, full, lightweight, and independent clients.
Syntax	*objServerObject*.**SetCustomerInfo** *customerId, customerKey* **Element** — **Description** *objServerObject* — The *ObjectServer* object to be acted on. *customerId* — A String containing the customer id. *customerKey* — A String containing the customer key.
Added to API	Prior to LNS Release 3.0.

SetFocus

Summary	This is a standard ActiveX control method used for visual controls. This method is not applicable for the Object Server.

SetLicenseInfo

Summary	This method is reserved for future use. Use the *SetCustomerInfo* method to set the licensing mode for your application.

SetLicenseInfoEx

Summary	This method is reserved for future use. Use the *SetCustomerInfo* method to set the licensing mode for your application.

ShowWhatsThis

Summary	This is a standard ActiveX control method which displays a selected topic in a help file. This method is not supported by the Object Server.

ZOrder

Summary	This is a standard ActiveX control method used for visual controls. This method is not applicable for the Object Server.

Properties

The *ObjectServer* object contains the following properties.

- *ActiveNetwork*
- *ActiveRemoteNI*
- *CausesValidation*
- *ClassId*
- *ComponentApps*
- *Container*
- *CurrentFormatLocale*
- *DatabasePath*
- *DragIcon*
- *DragMode*
- *Extensions*
- *Flags*
- *FormatLocales*
- *Height*
- *HelpContextID*
- *Index*
- *IsOpen*
- *LdrfLanguages*
- *LdrfCatalogPath*
- *Left*
- *Name*
- *NetworkInterfaces*
- *Networks*
- *Object*
- *Parent*
- *RemoteFlag*
- *RemoteNetworks*
- *ResourceLanguageId*
- *TabIndex*
- *TabStop*
- *Tag*
- *ToolTipText*
- *Top*
- *Version*
- *Visible*
- *VNINetworks*
- *WhatsThisHelpID*
- *Width*

ActiveNetwork

Summary	Contains the *Network* object currently being managed by the Object Server. When an Object Server is instantiated or initialized, it is assigned a network to manage; this is that Object Server's ActiveNetwork.
Availability	Local, full, lightweight, and independent clients.
Syntax	*networkObject* = *objServerObject*.**ActiveNetwork**

	Element	**Description**
	networkObject	The *Network* object designated as active.
	objServerObject	The *ObjectServer* object to be acted on.
Data Type	*Network* object.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ActiveRemoteNI

Summary	Specifies the network interface your remote Full client application will use to communicate with the OpenLNS Server. You must set this property before accessing the *Networks* collection object if you are using a remote Full client application. This property will only return a valid value after it has been explicitly set by an application. Remote Lightweight client applications do not need to set this property. When running as a Lightweight client, the Object Server's *NetworkInterfaces* collection object will contain only one *NetworkInterface* object, with the name "Internet". This "virtual" network interface is provided for backwards compatibility. The application may set the *ActiveRemoteNI* property to this *NetworkInterface* object. For more information on initializing remote OpenLNS applications, see the *OpenLNS Programmer's Guide.*
Availability	Full clients.
Syntax	*objServerObject*.**ActiveRemoteNI** = *niObject* **Element** — **Description** *niObject* — The *NetworkInterface* object to be designated as active. *objServerObject* — The *ObjectServer* object to be acted on.
Data Type	*Network* object.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

CausesValidation

Summary	This is a standard ActiveX control event used for visual controls. This event is not applicable for the Object Server.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ObjectServer* object in the *ConstClassIds* constant: **0 lcaClassIdObjectServer** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ComponentApps

Summary	Contains the *ComponentApps* collection object associated with the specified *ObjectServer* object. The *ComponentApps* collection is a list of OpenLNS plug-in commands that are associated with a particular object type. Note that all *LonMarkObject* objects contain a *ComponentApps* property. However, the behavior of this property is unspecified when accessed through a *LonMarkObject* object.
Availability	Local, full, and lightweight clients.
Syntax	*appsCollection* = *object*.**ComponentApps** **Element** — **Description** *appsCollection* — The *ComponentApps* collection to be returned. *object* — The *ObjectServer* object to be acted on.
Data Type	*ComponentApps* collection object.

Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Container

Summary	This is a standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

CurrentFormatLocale

Summary	Determines which *FormatLocale* object will be used by your application as the active *FormatLocale*. Each *FormatLocale* object contains a group of properties that reflect a particular geographical area's conventions for data display. These properties determine how data stored in the *FormattedValue* properties of *DataPoint* objects will be displayed when that *FormatLocale* is used by an application. The *CurrentFormatLocale* property determines which *FormatLocale* object will be used by your application as the active *FormatLocale*. The *FormatLocales* collection object contains all the *FormatLocale* objects that have been added to the Object Server. This includes four pre-defined *FormatLocale* objects. The default setting for this property is the "UserDefaultRegionalSettings" *FormatLocale* object, which uses collection index number 1. For a description of this and the other pre-defined Format Locales, see the *FormatLocales* collection object. **Note**: You can not write to this property, or write to the *FormatLocale* object assigned to this property after you have opened a network and formatted data with your application. Operations that will cause your application to format data include acquiring a *DataPoint* object, and reading or writing the value of a *ConfigProperty* or *NetworkVariable* object. If you write to the *CurrentFormatLocale* property after performing any of these operations, the **LCA, #122 lcaErrReadOnlyInContext** exception will be thrown. In addition, you can only access this property after you have *opened* the Object Server.
Availability	Local, full, lightweight, and independent clients.
Syntax	*objectServer.CurrentFormatLocale* = *formatLocale* **Element** — **Description** *objectServer* — The *ObjectServer* object being acted upon. *formatLocale* — The *FormatLocale* object to be used by the application.

Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

DatabasePath

Summary	Contains the path of the global OpenLNS database. You can only set the global database path before invoking the *Open* method for the *ObjectServer* object. The default location of the global database is **C:\LONWORKS\ObjectServer\GlobalDb**. OpenLNS does not apply a length restriction to the global database path.
Availability	Local, full, lightweight, and independent clients.
Syntax	*dbPath* = *object*.**DatabasePath** **Element** — **Description** *dbPath* — The full path of the global OpenLNS database. *object* — The *ObjectServer* object.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DragIcon

Summary	This is a standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

DragMode

Summary	This is a standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *LonMarkObject*. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer.
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object.Extensions* **Element** — **Description**

	extensionsColl	The *Extensions* collection object.
	object	The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Flags

Summary	Specifies a series of global flags that determine how OpenLNS will behave with the client application's processes, including the type of network variable update used, post-connection update state, remote application transfer state and the remote-IP mode used by the OpenLNS Server.	
Availability	Local, full, lightweight, and independent clients.	
Syntax	*flagsValue = osObject.Flags*	
	Element	**Description**
	osObject	The *ObjectServer* object to be acted on.
	flagsValue	The *Flags* value as an integer. The valid flag options for this element, which are specified in the *ConstGlobalFlags* constant, are as follows: **1 lcaFlagsStringUpdate** Causes the OpenLNS Server to use the *OnNetworkVariableStringUpdate* event, which passes a String type where *OnNetworkVariableUpdate* passes a Variant. This flag is not required for Visual Basic or Visual C++. Some development environments do not handle Object types in event parameters. If you are programming in such an environment, use this flag. **2 lcaFlagsConnectionUpdatesOnLine** Specifies whether devices will normally be left online while their connections are updated. This behavior can be explicitly set for an individual device using the *ConnectionUpdateType* property. The default OpenLNS Server behavior is to take devices off line while

		connections are updated. You should not set this value to True because your application may then process or send network variable updates using inconsitent network variable configuration. This could lead to misinterpretation of network variable updates on this or another device. **16 lcaFlagsUseTCP** Indicates that a remote client (see *RemoteFlag*) will access the OpenLNS Server via TCP/IP. This flag should not be ORed with the **lcaFlagsUseNSI (32)** flag. **32 lcaFlagsUseNSI** Indicates that a remote client (see *RemoteFlag*) will access the OpenLNS Server via an NSI over a LONWORKS network. This flag should not be ORed with the **lcaFlagsUseTCP (16)** flag. **1024 lcaFlagsDirectCallback** Enables callback routines (event handlers) to be executed from an internal LNS thread, as opposed to the thread that instantiated the Object Server. This essentially turns your application into a multi-threaded application. Executing event handlers from non-client threads in this fashion is often more efficient. By default, this flag is not set. However, the ability to execute handler routines is not supported by all development tools. It is supported by Visual C++ and Microsoft Visual Studio .NET, but not by Visual Basic. In addition, there are several important programming considerations you need to be aware of when using multiple threads within an OpenLNS application. For more information on multi-threading with OpenLNS applications, see the *OpenLNS Programmer's Guide*. **2048 lcaFlagsManualNsdUpgrade** Indicates whether client's *Network*

	Service Device will be automatically upgraded whenever an upgrade is required. If this flag is not set, OpenLNS will perform the upgrades automatically. The flag is not set by default. If you set this flag, you will need to upgrade your client's *NetworkServiceDevices* manually when any of the following situations occur: • A new version of OpenLNS is installed which changes the Network Service Device's program interface. For example, OpenLNS 3.0 added monitor sets to the program interface, and therefore required a Network Service Device upgrade. • Switching from a layer 2 MIP, or engineered mode, to a Layer 5 MIP. • Switching between certain layer 5 MIPs with different capabilities. For more information on this, see the *Upgrading a Network Service Device* section in the *OpenLNS Programmer's Guide*. Multiple options may be specified by logically OR'ing the individual flag values together. You should only write to this property before you have *opened* the *ObjectServer*.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

FormatLocales

Summary	Contains a series of properties that reflect a particular geographical area's conventions for data display. These properties determine how data stored in the *FormattedValue* properties of *DataPoint* objects will be displayed when that *FormatLocale* is used by an application. The *FormatLocales* property contains the collection of *FormatLocale* objects that have been added to the *Object Server*. You can use this property to access the *FormatLocales* collection. The *FormatLocales* collection contains four

	pre-defined *FormatLocale* objects, and you can create custom *FormatLocale* objects with the *Add* method. For more information, see the *FormatLocales* collection object. **Note**: You can only access this property after you have *opened* the Object Server.
Availability	Local, full, lightweight, and independent clients.
Syntax	*flCollection = objectServer.FormatLocales* **Element** — **Description** *flCollection* — The *FormatLocales* collection object returned. *objectServer* — The *ObjectServer* object being acted upon.
Data Type	*FormatLocales* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Height

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

HelpContextId

Summary	This is a standard ActiveX control property used for context help. This property is not applicable for the Object Server.

Index

Summary	Contains a number identifying an Object Server control in a control array.
Availability	Local, full, lightweight, and independent clients.
Syntax	*indexValue = osObject*.**Index** **Element** — **Description** *osObject* — Te *ObjectServer* object to be acted on. *indexValue* — The identifier of the item in the array.
Data Type	Integer.
Read/Write	Read only

IsOpen

Summary	Indicates whether the specified *ObjectServer* object is currently open.
Availability	Local, full, lightweight, and independent clients.
Syntax	*isOpenFlag = Object*.**IsOpe**

	Element	**Description**
	isOpenFlag	Boolean value.
	Object	A Boolean value indicating whether the *ObjectServer* object is currently open. **TRUE**. The *ObjectServer* object is currently open. **FALSE**. The *ObjectServer* object is currently closed.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

LdrfLanguages

Summary	Contains the *LdrfLanguages* collection representing the languages known by OpenLNS. You can only access this property after you have *opened* the Object Server.	
Availability	Local, full, lightweight, and independent clients.	
Syntax	*ldrfLang = osObject.LdrfLanguages*	
	Element	**Description**
	ldrfLang	The *LdrfLanguages* collection.
	osObject	The *ObjectServer* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

LdrfCatalogPath

Summary	Obtains the full path for the location of the LonMark device resource file catalog (**ldrf.cat**). You should not modify this path because the resource file catalog should always be stored in the **LonWorks/Types** folder. The *LdrfCatalogPath* property points this path by default; therefore, you should not write to this property under any circumstances. See the *LonMark Resource File Catalog Help* and the *Device Resource File Developers Guide* Setting this property in the *ObjectServer* object overrides the value in the *System* object. This must be done by a Local client application before you call the *ObjectServer* object's *Open* method. However, you should not write to this property. If you do write to the *LdrfCatalogPath* property, leave the standard resource files in the **LonWorks/Types** folder, and then create a copy of the standard resource files in the new folder referenced by the *LdrfCatalogPath* property
Availability	Local, full, lightweight, and independent clients.
Syntax	*pathName = object.LdrfCatalogPath* **Element** — **Description** *object* — The *ObjectServer* object to be acted on. *pathName* — The location of the LDRF catalog, which contains the locations of the standard and user-defined resource files, and the associated files that are required for data formatting. The *pathName* cannot include the semi-colon character (;).
Data Type	String.
Read/Write	Read and write for Local client applications. Read only for Full and Lightweight client applications.
Added to API	LNS Release 3.0.

Left

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.

Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NetworkInterfaces

Summary	Returns the *NetworkInterfaces* collection object representing all of the network interfaces in the system registered on the Object Server's computer.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*niCollection* = *objectServerObject*.*NetworkInterfaces* **Element** — **Description** *niCollection* — The returned *NetworkInterfaces* collection object. *objectServerObject* — The *ObjectServer* object to be acted upon.
Data Type	*NetworkInterfaces* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Networks

Summary	A collection of all the local networks.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*networksCollection* = *objectServerObject*.*Networks* **Element** — **Description** *networksCollection* — The returned *Networks* collection. *objectServerObject* — The *ObjectServer* object to be acted on.

Data Type	*Networks* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Object

Summary	Allows invocation of a control's method or property that has the same name as a method or property automatically extended by the control container. See your development tool's help for more information on this property.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*propValue* = *osObject*.**Object**.*property* *osObject*.**Object**.*method* **Element** — **Description** *propValue* — The returned property value. *osObject* — The *ObjectServer* object to be acted on. *property* — The property to be accessed. *method* — The method to be invoked
Data Type	Object.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.

Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

RemoteFlag

Summary	Specifies whether the network will be opened locally or remotely. This property must be set prior to opening the OpenLNS Server. See the *Open* method for more information. The *RemoteFlag* property must be set to True for applications that are to be distributed as LNS Remote distributables. Otherwise, the **LCA, #72 lcaErrWrongServerDll** exception will be thrown when you open the OpenLNS Object Server. You will establish whether an application is to be distributed as an OpenLNS Remote or an OpenLNS Complete redistributable when you create the redistributable with the OpenLNS SDK.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *remoteFlag* — A Boolean value indicating whether the network will be opened locally or remotely. **TRUE**. The network will be opened remotely. When this property is True, the Object Server's *Flags* property may be used to specify the transport protocol to be used by the OpenLNS application. The relevant flags are **lcaFlagsUseTCP** or **lcaFlagsUseNSI**. **FALSE**. The network will be opened locally. *osObject* — The *ObjectServer* control.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

RemoteNetworks

Summary	Contains a collection of all remote full client networks that have been opened by an application running on this client's computer. The OpenLNS Object Server views the networks in this collection as local networks. To view this collection, you must set the *RemoteFlag* property to False.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *remNetworksColl* — The *Networks* collection to be returned. *osObject* — The *ObjectServer* object to be acted on.
Data Type	*Networks* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ResourceLanguageId

Summary	Controls which language should be used when displaying descriptive type information stored in device resource files. This property can contain one or more language codes that identify the language that should be used when displaying type information stored in resource files. When you pass multiple language codes to this property, they must be comma-separated, as in the following example: "frc, enz, rus." The language files will be searched for in the order that the codes are supplied. For example, if this property is set to "frc,enz,rus", OpenLNS will first look for the .frc (French Canadian) language files, and then the .enz and .rus language files when it searches the resource files. If none of the values in the list are found, the default of "enu" (U.S. English) will be used. All language codes are three characters long. See the *LonMark Device Resource File Developer's Guide* for information on resource language IDs and a partial list of the language codes you can pass to this property. Setting this property in the *ObjectServer* object overrides the value in the *System* object. It also allows this property to be set when the *System* object is not available (i.e. when performing standalone monitor and control). The default value for this property is "enu" (U.S. English). You can only access this property after you have *opened* the Object Server. **Note**: The *FormatLocale* object contains a *LanguageId* property. This property determines what language file OpenLNS will use to display data stored in the *FormattedValue* properties of *DataPoint* objects when the *FormatLocale* is being used by an application. See the *FormatLocale* object for more information on this.
Availability	Local, full, lightweight, and independent clients.
Syntax	*languageId = systemObject.ResourceLanguageId* **Element** — **Description** *languageId* — A comma-separated list of one or more language codes identifying the languages that should be used when displaying type information. *systemObject* — The *System* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

TabIndex

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

TabStop

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

Tag

Summary	Contains any extra data needed for your program. The data stored in this property is not used by the Object Server, and can be used for any purpose by the OpenLNS application.
Availability	Local, full, lightweight, and independent clients.
Syntax	*tagValue* = *Object*.**Tag** **Element** — **Description** *Object* — The *ObjectServer* object to be acted on. *tagValue* — The tag associated with the *ObjectServer* object.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ToolTipText

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

Top

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

Version

Summary	Returns the version of the OpenLNS Object Server ActiveX control that is being used.
Availability	Local, full, lightweight, and independent clients.
Syntax	*version* = *osObject*.**Version** **Element** — **Description** *version* — The returned version number as a string with the format M.NN, where M is the major version number and NN is the minor version number (for example,

		"4.00"). The version number is the same value as that returned in the About Box
	osObject	The *ObjectServer* object to be acted on.
Data Type	String.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Visible

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

VNINetworks

Summary	Contains the *Networks* collection containing all the *Network* objects which can be opened in *server-independent* mode. This property contains all networks which can be opened in server-independent mode. Only local clients and remote full clients can be opened in this manner. You should only use the *OpenIndependent* method to open a *Network* object fetched from this collection. If you want to open the network in server-dependent mode (with the *Network* object's *Open* method), fetch the *Network* object from the *Networks* collection (if using a local client or a remote full client for the first time from this computer) or the *RemoteNetworks* collection (if reopening the network as a remote full client).	
Availability	Local, full, and independent clients.	
Syntax	*vniColl = osObject.VNINetworks*	
	Element	**Description**
	vniColl	The *Networks* collection object to be returned.
	detailInfoObject	The *ObjectServer* object to be acted on.
Data Type	*Networks* collection object.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

WhatsThisHelpID

Summary	This property returns or sets an associated context number for an object. You can use this property to provide context-sensitive help for your application for 32-bit Windows operating systems. This property is not supported by the Object Server.

Width

Summary	A standard ActiveX control property used for visual controls. This property is not applicable for the Object Server.

Events

The *ObjectServer* object contains the following events:

- *DragDrop*
- *DragOver*
- *GotFocus*
- *LostFocus*
- *OnAttachment*
- *OnChangeEvent*
- *OnCommission*
- *OnDbConversionEvent*
- *OnDbValidationEvent*
- *OnIncomingSessionEvent*
- *OnLonMarkObjectStatusChangeEvent*
- *OnMissedEvent*
- *OnMsgMonitorPointErrorEvent*
- *OnMsgMonitorPointEvent*
- *OnMsgMonitorPointUpdateEvent*
- *OnNetworkServiceDeviceResetNew*
- *OnNetworkVariableStringUpdate*
- *OnNetworkVariableUpdate*
- *OnNodeConnChangeEvent*
- *OnNodeIntfChangeEvent*
- *OnNvMonitorPointErrorEvent*
- *OnNvMonitorPointEvent*
- *OnNvMonitorPointUpdateEvent*
- *OnNVUpdateError*
- *OnSessionChangeEvent*
- *OnSystemMgmtModeChangeEvent*
- *OnSystemNssIdle*
- *OnSystemServicePin*
- *Validate*

DragDrop

Summary	A standard ActiveX control event used for visual controls. This event is not applicable for the Object Server.

DragOver

Summary	A standard ActiveX control event used for visual controls. This event is not applicable for the Object Server.

GotFocus

Summary	A standard ActiveX control event used for visual controls. This event is not applicable for the Object Server.

LostFocus

Summary	A standard ActiveX control event used for visual controls. This event is not applicable for the Object Server.

OnAttachmentEvent

Summary	Indicates that the attachment status has changed for an *AppDevice* or *Router*. This event is generated whenever a device or router is attached to the network. Once an *AppDevice* or *Router* has been attached, it is monitored via periodic pinging to ensure that it remains attached. If the ping fails, indicating that the device or router is no longer attached, this event will be generated to indicate this. **Note**: OpenLNS will check to make sure that the ping failure is due to a failure on the device or router being pinged, and not because of a failure on another router on the communications path between the OpenLNS Object Server and the router or device in question. The rate at which a device or router is pinged is determined by its *PingClass* property. This event is fired when changes are made to the physical network. The *OnChangeEvent* is fired when those changes are made to the OpenLNS database. You can subscribe or unsubscribe your application to these events using the *System* object's *BeginAttachmentEvent* and *EndAttachmentEvent* methods. For more information on the *OnAttachment* event, see the *Testing Devices and Detecting Device Failures* section in the *OpenLNS Programmer's Guide.*

Syntax	**OnAttachment**(*networkHandle* As Long, *systemHandle* As Long, *objectType* as Integer, *isAttached* As Boolean, *stateFailure* as Boolean, *objectHandle* As Long)

Element	Description
networkHandle	Handle of the network in which the attached or unattached device or router resides.
systemHandle	The Handle of the system in which the attached or unattached device or router resides.
objectType	The type of object that was attached or unattached. The valid values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**
isAttached	A Boolean value that indicates whether the device or router is being attached or unattached. **TRUE**. The device or router is currently attached to the network, and in the proper state. **FALSE**. The device or router is not attached to the network, or there is a state failure on the device or router. You can use the *stateFailure* element to determine if there is a state failure on the device.
stateFailure	Indicates whether there is a state failure on the object. A state failure may occur when a device or router has been installed, and the state of the device or router (reported by the *State* property) is anything other than **lcaStateCnfgOnline**, **lcaStateSoftOffline**, or **lcaStateOfflineBypass**. A state failure may also occur if the subnet/node address assigned to a device or router in the OpenLNS database is incorrect. You can check the subnet/node address of a device or router by reading the *SubnetId*

		and *NodeId* properties. If the state is **lcaStateUncnfg** or **lcaStateCnfgOffline**, you can resolve this by recomissioning the device or router with the *Commission* method. If the state of an application device is **lcaStateNoApplUncnfg**, and that device is a Neuron hosted device, you must reload the application with the *Load* method. After either operation, the state of the device or router will be **lcaStateSoftOffline**. You may want to change the *State* property of the device or router to **lcaStateSoftOnline** after the load or commission succeeds.
	ObjectHandle	The *Handle* of the attached or unattached object.
Added to API	LNS Release 3.0.	

OnChangeEvent

Summary	Indicates that an object has been modified in the OpenLNS database. This event may be used by applications to maintain lists of objects, such as those that might appear on a user interface. You can use the *System* object's *BeginChangeEvent* and *EndChangeEvent* methods to subscribe and unsubscribe your application to this event. When you register for the event with the *BeginChangeEvent* method, you will select an object type(s) to receive events for. The *OnChangeEvent* event will then be fired whenever changes are made to objects of that type(s) in the OpenLNS database. You can use the *OnAttachmentEvent* to be notified of attachments to or detachments from the physical network.
Syntax	**OnChangeEvent**(*networkHandle* As Long, *systemHandle* As Long, *objectType* as Integer, *changeType* As Integer, *objectHandle* As Long)

Element	**Description**
networkHandle	Handle of the network in which the changed object resides.
systemHandle	Handle of the system in which the changed object resides.
objectType	The type of object that changed. The enumerated values for this element, which are contained in the *ConstChangeEventTypes*

		constant, are as follows: **0 lcaChangeEventAppDevices** **1 lcaChangeEventChannels** **2 lcaChangeEventRouters** **3 lcaChangeEventSubnets** **4 lcaChangeEventNsis** **5 lcaChangeEventSubsystems** **6 lcaChangeEventDeviceTemplates** **7 lcaChangeEventDomains** **8 lcaChangeEventConnectDescTemplates** **9 lcaChangeEventExtensions**
	changeType	Indicates how the object changed. The enumerated values for this element, which are contained in the *ConstObjectChangeTypes* constant, are as follows: **0 lcaObjectCreate** An object was created. **1 lcaObjectDelete** An object was deleted. **4 lcaObjectRegistered** An object was registered. **5 lcaObjectDeregistered** An object was deregistered. **6 lcaObjectReplaced** An object was replaced. **7 lcaObjectUpgraded** An object was upgraded. **8 lcaObjectRenamed** An object was renamed. **10 lcaObjectDescriptionModified** Reserved for future use. **11 lcaObjectSubsystemMemebershipModifed** An *AppDevice* or *Router* was added or removed from a subsystem.
	objectHandle	The handle of the changed object. This value will be **0** when the *objectType* is **lcaChangeEventDeviceTemplates**.
Added to API	Prior to LNS Release 3.0.	

OnCommission

Summary	Indicates that a device's commission status has changed. You can register your application for the OnCommission event by invoking the *BeginCommissionEvents* method. You can subscribe or unsubscribe your application to this event by calling the *System* object's *BeginCommissionEvent* and *EndCommissionEvent* methods. When you create a network, and then change the network management mode of the OpenLNS Object Server from **lcaMgmtModeDeferConfigUpdates (1)** to **lcaMgmtModePropagateConfigUpdates (0)**, the NSI is commissioned automatically. However, if you registered for *OnCommissionEvent* while the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**, your application will not receive an event indicating this. In this case, you can check the commission status of the NSI after the network management mode has been set to **lcaMgmtModePropagateConfigUpdates (0** by reading the *CommissionStatus* of the *System* object's *NetworkServiceDevice*. You can change the Object Server's network management mode by writing to the *MgmtMode* property.
Syntax	**OnCommission**(*networkHandle* As Long, *systemHandle* As Long, *objectType* as Integer, *commissionStatus* As Integer, *channelHandle* as Long, *objectHandle* As Long) **Element** — **Description** *networkHandle* — Handle of the network in which the commissioned object resides. *systemHandle* — Handle of the system in which the commissioned object resides. *objectType* — The type of object that was commissioned. The enumerated values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent** *commissionStatus* — The status of the commissioning updates. The commission status changes to **lcaCommissionUpdatesPending (1)** when database changes are made that affect a device's configuration. , or they change to **lcaCommissionUpdatesFailed (2)** when there is a failure to propagate

		changes. The enumerated values for this element, which are contained in the *ConstCommissionStatus* constant, are as follows: **0 lcaCommissionUpdatesCurrent** No outstanding commission updates are pending. The commission status changes **lcaCommissionUpdatesCurrent (0)** when database changes have been successfully propagated to the device **1 lcaCommissionUpdatesPending** Commission updates are currently pending, or in progress. When database changes are made that affect a device's configuration, this value represents the commission status of the device. The commission status changes to **lcaCommissionUpdatesPending (1)** when database changes that affect a device's configuration are made. **2 lcaCommissionUpdatesFailed** Commission updates are currently pending or in progress, and the most recent update attempt failed. The commission status will be changed to **lcaCommissionUpdatesFailed (2)** when database changes fail to be propagated.
	channelHandle	The channel on which the object resides
	objectHandle	*The Handle* of the commissioned object
Added to API	Prior to LNS Release 3.0.	

OnDbConversionEvent

Summary	Provides a client application the progress of an OpenLNS database conversion. When an OpenLNS database is opened using an application written with a newer version of OpenLNS, the database will be automatically upgraded to the new version of OpenLNS. For example, if an application running on OpenLNS calls the *Open* method to open a network using an LNS 3.20 database, the database would be upgraded automatically. If an application running on OpenLNS calls the *Add* method to create a network and specifies an existing LNS 4.0 database as the new network's database, the LNS 3.20 database would also be upgraded automatically. The OpenLNS database conversion is a three-step process that upgrades the system database, the network database, and the global database. This event is triggered once for each stage of the conversion, and once when the conversion is complete. After a database has been upgraded, it cannot be opened with an older version of LNS (an application running on LNS 3.20 cannot open an OpenLNS database).
Syntax	**OnDbConversionEvent**(*dbName* As String, *stage* as Integer) **Element** — **Description** *dbName* — The name of the OpenLNS database being upgraded. *stage* — The stage of the database conversion. The possible values for this element, which are contained in the *ConstDbConversionStage* constant, are as follows: **0 lcaConversionDone** The conversion is complete. **1 lcaGlobalDb** The global network database is being converted. **2 lcaNetworkDb** The network database is being converted. **3 lcaSystemDb** The system database is being converted.
Added to API	LNS Release 3.0.

OnDbValidationEvent

Summary	Provides a client application the progress of database validation. A database validation can be initiated with the *Validate* method. Depending on the size of a network database, a database validation can take a considerable amount of time to complete. You can use the information returned by this event to develop an idea of how long the database validation being performed will take. This event will be fired many times throughout the database validation, including each time the validation progresses to a new phase, and each time the validation progresses to a new step within that phase. Alternatively, you can also receive updates for a database validation directly through an *ILcaProgressListener* interface object. In order to do so, you must specify the *progressCallback* element when you call the *Validate* method. The number of phases and the names of the steps and phases involved in a database validation may differ from datatabase to database, and from release to release. You can use them to display a progress bar with your application, but you should not expect this event to return for the same number of phases or steps each time you perform a database validation.
Syntax	**OnDbValidationEvent** (*totalPercentage* as Long, *thisPhasePercentage* as Long, *thisPhaseNumber* as Long, *totalPhases* as Long, *thisPhaseName* as String, *thisStepName* as String)

Element	**Description**
totalPercentage	The percentage of the database validation that has been completed. This element has a range of 0–100.
thisPhasePercentage	The percentage of the current phase of the database validation that has been completed. You can use the *thisPhaseName* element to determine which phase of the database validation is currently being performed. This element has a range of 0–100.
thisPhaseNumber	The number of the current phase. This element has a range of 0–2,147,483,647.
totalPhases	The total number of phases to be performed during the database validation. This element has a range of 0–4,292,967,296.
thisPhaseName	The name of the phase of the

		database validation that is currently being performed. The phase name will be returned as a string of up to 230 characters.
	thisStepName	The step that is currently being performed. The step name will be returned as a string of up to 230 characters. Generally, this will be the name of the object in the database that is currently being validated.
Added to API	LNS Release 3.20.	

OnIncomingSessionEvent

Summary	An event that is generated whenever a request for connection to the OpenLNS Server is received, as long as the application has registered for uplink session event handling with the *BeginIncomingSessionEvents* method. After an uplink session request has been received and this event has been fired, you can use the *AcceptIncomingSession* method to accept or reject the request. When you call the *AcceptIncomingSession* method, you use the *tag* element passed to this event to identify the xDriver session. The other elements can be used to open the network after the incoming session is accepted. Do not open the network within the event handler. Instead, signal your main thread to open the network by posting a message, or by using a timer. Once you have accepted a session, you can open the network identified by the *netName* element and perform whatever tasks are required. You can also use the *OnSessionChangeEvent* event to monitor the state of the xDriver session. You can disable the *OnIncomingSessionEvent* event with the *EndIncomingSessionEvents* method. These events only apply to clients that are using the OpenLDV xDriver to connect to remote network interface (RNI) devices. For an overview of the OpenLDV xDriver, see the *OpenLNS Network Interface Drivers* section, of the *OpenLNS Programmer's Guide*.

Syntax	*OnIncomingSessionEvent*(xDriverProfileName, netName, intfName, tag) **Element** — **Description** *xDriverProfileName* — A String identifying the Profile that is using the TCP listener port this session came in on. This may be useful in an application that registers for uplink session event handling with multiple xDriver Profiles. *netName* — A String that represents the OpenLNS network name of the network that has requested the uplink session. *intfName* — A String that represents the network interface name of the network that has requested the uplink session. *tag* — This value must be used when the *AcceptIncomingSession* method is called to accept or reject the uplink session.
Added to API	LNS Release 3.06.

OnLonMarkObjectStatusChange

Summary	Once an application registers for this event, it will be fired each time an OpenLNS application changes the status of a *LonMarkObject* by writing to the object's *Request* property. You can enable this event for your application by invoking the *BeginLonMarkObjectStatusChangeEvent* method. You can disable the event by invoking the *EndLonMarkObjectStatusChangeEvent* method. You can write to the *Request* property of the *LonMarkObject* object to change the current functionality of a LonMark object, or to update the information contained in the *LonMarkObject*. These changes would cause the *OnLonMarkObjectStatusChangeEvent* event to be fired. You can read the *Status* property to determine the current status of a *LonMarkObject*. Note that this event will only be fired when an LNS application changes the status of a *LonMarkObject*. It will not be fired if the status is changed directly by the device, or if you explicitly write to the device's SNVT_object_request network variable.

Syntax	**OnLonMarkObjectStatusChangeEvent**(*networkHandle*, *systemHandle*, *deviceHandle*, *LonMarkObjectIndex*) **Element** — **Description** *networkHandle* — The handle of the *Network* containing the modified *LonMarkObject*. *systemHandle* — The handle of the *System* containing the modified *LonMarkObject*. *deviceHandle* — The handle of the *AppDevice* using the *Interface* that contains the modified *LonMarkObject*. *LonMarkObjectIndex* — The device index number of the modified *LonMarkObject*. This value index is stored in the modified LonMarkObject's *Index* property.
Added to API	LNS Release 3.20.

OnMissedEvent

Summary	Indicates that one or more generated events were not received by the applications that subscribed to them. You can enable this event with the *BeginMissedEvent* method. You can disable this event with the *EndMissedEvent* method.
Syntax	**OnMissedEvent**(*networkHandle* As Long, *systemHandle* As Long, *isUnrecoverable* as Boolean, *numMissedEvents* As Long) **Element** — **Description** *networkHandle* — Handle of the network in which the object resides. *systemHandle* — Handle of the system in which the object resides. *isUnrecoverable* — Indicates whether the event is recoverable. *numMissedEvents* — Indicates how many events were missed.
Added to API	Prior to LNS Release 3.0.

OnMsgMonitorPointErrorEvent

Summary	An event that is generated whenever there is a write failure on a message monitor point. This event is also used to signify asynchronous unsuccessful completion code events. If you are using a programming environment which supports

	multi-threading (such as Visual C++), you can receive update events on a separate thread by creating an object which implements the *ILcaNvMonitorPointListener* .
Syntax	*OnMsgMonitorPointErrorEvent*(*msgMonitorPoint* As Object, *ErrorType* as Integer) **Element** — **Description** *msgMonitorPoint* — The *MsgMonitorPoint* object that reported an error. *ErrorType* — The error type for the event. The possible values for this element, which are contained in the *ConstMonitorEventType* constant, are as follows: **0 lcaMonitorEventTypeNull** This value is not used. **1 lcaMonitorEventTypeQuit** This value is not used. **2 lcaMonitorEventTypeAdd** This value is not used. **3 lcaMonitorEventTypeRemove** This value is not used. **4 lcaMonitorEventTypeMsCreate** This value is not used. **5 lcaMonitorEventTypeMsDelete** This value is not used. **6 lcaMonitorEventTypeMsChange** This value is not used. **7 lcaMonitorEventTypeMsError** This value is not used. **8 lcaMonitorEventTypeNvCreate** A network variable monitor point has been created. **9 lcaMonitorEventTypeNvDelete** A network variable monitor point has been removed. **10 lcaMonitorEventTypeNvChange** This value is not used. **11 lcaMonitorEventTypeNvError** A network variable monitor point

	has returned an error. See the *OnNvMonitorPointErrorEvent* event. **12 lcaMonitorEventTypeNvUpdate** A network variable monitor point has received an update. See the *OnNvMonitorPointUpdateEvent* event. **13 lcaMonitorEventTypeNvComplete** A completion code has returned for the monitor point. **14 lcaMonitorEventTypeMsgCreate** A message monitor point has been created. **15 lcaMonitorEventTypeMsgDelete** A message monitor point has been removed. **16 lcaMonitorEventTypeMsgChange** This value is not used. **17 lcaMonitorEventTypeMsgError** A message monitor point has returned an error. See the *OnMsgMonitorPointErrorEvent* event. **18 lcaMonitorEventTypeMsgUpdate** A message monitor point has received an update. See the *OnMsgMonitorPointUpdateEvent* event. **19 lcaMonitorEventTypeMsgRequest** A message monitor point has received a request message. **20 lcaMonitorEventTypeMsgResponse** A message monitor point has received a reponse message. **21 lcaMonitorEventTypeMsgComplete** This value is not used.
Added to API	LNS Release 3.0.

OnMsgMonitorPointEvent

Summary	An event that is generated whenever a message monitor point is *created* or *removed* from a permanent monitor set. The events are only fired in this case if the monitor set is open. In addition, when you *close* or *open* a permanent monitor set, this event will be fired for each message monitor point in the set, indicating that the monitor points have been removed from or created in memory. This event is also fired the first time you *enable* a message monitor point in a temporary monitor set after creating the temporary monitor set. It will also be fired when you remove a message monitor point from a temporary monitor set, as long as that monitor point had been enabled. This event is also used to signify asynchronous successful completion code events for a message monitor point.
Syntax	*OnMsgMonitorPointEvent*(*msgMonitorPoint* As Object, *EventType* as Integer) **Element** — **Description** *msgMonitorPoint* — The *MsgMonitorPoint* object that was created or shut down. *EventType* — The event code for the event. The possible values for this element, which are contained in the *ConstMonitorEventType* constant, are as follows: **0 lcaMonitorEventTypeNull** This value is not used. **1 lcaMonitorEventTypeQuit** This value is not used. **2 lcaMonitorEventTypeAdd** This value is not used. **3 lcaMonitorEventTypeRemove** This value is not used. **4 lcaMonitorEventTypeMsCreate** This value is not used. **5 lcaMonitorEventTypeMsDelete** This value is not used. **6 lcaMonitorEventTypeMsChange** This value is not used. **7 lcaMonitorEventTypeMsError** This value is not used.

		8 lcaMonitorEventTypeNvCreate A network variable monitor point has been created. **9 lcaMonitorEventTypeNvDelete** A network variable monitor point has been removed. **10 lcaMonitorEventTypeNvChange** This value is not used. **11 lcaMonitorEventTypeNvError** A network variable monitor point has returned an error. See the *OnNvMonitorPointErrorEvent* event. **12 lcaMonitorEventTypeNvUpdate** A network variable monitor point has received an update. See the *OnNvMonitorPointUpdateEvent* event. **13 lcaMonitorEventTypeNvComplete** A completion code has returned for the monitor point. **14 lcaMonitorEventTypeMsgCreate** A message monitor point has been created. **15 lcaMonitorEventTypeMsgDelete** A message monitor point has been removed. **16 lcaMonitorEventTypeMsgChange** This value is not used. **17 lcaMonitorEventTypeMsgError** A message monitor point has returned an error. See the *OnMsgMonitorPointErrorEvent* event. **18 lcaMonitorEventTypeMsgUpdate** A message monitor point has received an update. See the *OnMsgMonitorPointUpdateEvent* event. **19 lcaMonitorEventTypeMsgRequest** A message monitor point has

		received a request message. **20 lcaMonitorEventTypeMsgResponse** A message monitor point has received a reponse message. **21 lcaMonitorEventTypeMsgComplete** This value is not used.
Added to API	LNS Release 3.0.	

OnMsgMonitorPointUpdateEvent

Summary	An event that is generated whenever a message tag monitor point update is received. The *OnMsgMonitorPointEvent* event is used to signify asynchronous successful completion code events for a message monitor point. The *OnMsgMonitorPointErrorEvent* is used to signify asynchronous unsuccessful completion code events. If you are using a programming environment which supports multi-threading (such as Visual C++), you can receive update events on a separate thread by creating an object which implements the *ILcaMsgMonitorPointListener* interface.
Syntax	*OnMsgMonitorPointUpdateEvent*(*msgMonitorPoint* As Object, *UpdateType* as Integer, *InputDp* as Object, *OutputDp* as Object, *Src* as Object) **Element** — **Description** *msgMonitorPoint* — The *MsgMonitorPoint* object that received an update. *UpdateType* — The update type. *InputDp* — A *DataPoint* object containing the received value. The *AutoRead* and *AutoWrite* properties of this element are set to FALSE. *OutputDp* — A *DataPoint* object which allows a response to be sent if the UpdateType is Request. The *AutoWrite* of this element is set to TRUE; therefore, this *DataPoint* object will be sent as a response when it is updated. *Src* — A *SourceAddress* object indicating the source device of the update.

Added to API	LNS Release 3.0.

OnNetworkServiceDeviceResetNew

Summary	Indicates that the local network interface has reset. This event is identical to the old *OnNetworkServiceDeviceReset* event, but it includes information regarding the network service device's network, system, and object handles. Lightweight Client applications cannot receive the *OnNetworkServiceDeviceResetNew* event. You can enable this event for your application by calling the *BeginResetEvent* method. You can disable this event by calling the *EndResetEvent* method.
Syntax	**OnNetworkServiceDeviceResetNew**(*nsdObject* As Object, *networkHandle* as Long, *systemHandle* as Long, *objectHandle* as Long) **Element** — **Description** *nsdObject* — The *NetworkServiceDevice* object that reset. *networkhandle* — The handle of the network in which the *NetworkServiceDevice* object resides. *systemHandle* — The handle of the system in which the *NetworkServiceDevice* object resides. *objectHandle* — The object handle of the *NetworkServiceDevice* object being reset.
Added to API	Prior to LNS Release 3.0.

OnNetworkVariableStringUpdate

Summary	Indicates that a network variable value has been updated. You can register for this event by setting the *DsMonitorTag* property to a non-zero value. This event was created for use with programming environments that do not support passing Object types to event parameters. It is identical to the *OnNetworkVariableUpdate* event except that the *value* parameter returns a String rather than a Variant. If you want to use this event rather than the *OnNetworkVariableUpdate* event, set the *lcaFlagsStringUpdate* flag in the *ObjectServer* object's *Flags* property to **1**. If you are using Visual C++, the *OnNetworkVariableStringUpdate* event will not be fired in

	your application's main thread. Instead, a separate thread will be used for this event. When using the separate thread for these events, you should perform as little processing as possible within your event handler. However, you can use the *PostMessage*() or *PostMessageThread*() functions of the Win API to defer these events to the application's main thread. Consult the Windows documentation for more information on these functions. For more information on multi-threading and OpenLNS, see the *Multi-Threading and OpenLNS Applications* section in the *OpenLNS Programmer's Guide*.
Syntax	**OnNetworkVariableStringUpdate**(*monitorTag* As Long, *value* as String, *dataUpdateInfo* As Object) **Element** — **Description** *monitortag* — The monitor tag value assigned to the network variable. *value* — Current data value of the network variable. *dataUpdateInfo* — The *DataValue* object for this update.
Added to API	Prior to LNS Release 3.0.

OnNetworkVariableUpdate

Summary	Indicates that a network variable value has been updated. You can register for this event by setting the *DsMonitorTag* property to a non-zero value. If you are using a programming environment that cannot handle Object types being passed as an event parameter, you should use the *OnNetworkVariableStringUpdate* event instead of this one. To use that event, set the *ObjectServer* object's *Flags* property to 1. Note that if you are using Visual C++, the *OnNetworkVariableUpdate* event will not be fired in your application's main thread. A separate thread will be used for this event. When using the separate thread for these events, Echelon recommends that you perform as little processing as possible within your event handler. However, you can use the PostMessage() or PostMessageThread() functions of the Win API to defer these events to the application's main thread. Consult the Windows documentation for more information on these functions. For more information on multi-threading and OpenLNS, see the *Multi-Threading and OpenLNS Applications* section in the *OpenLNS Programmer's Guide*.

Syntax	*OnNetworkVariableUpdate* (*monitorTag* As Long, *value* as Variant, *dataUpdateInfo* As Object) **Element** — **Description** *monitortag* — The monitor tag value assigned to the network variable. *value* — Current data value of the network variable. *dataUpdateInfo* — The *DataValue* object for this update.
Added to API	Prior to LNS Release 3.0.

OnNodeConnChangeEvent

Summary	This event is generated whenever a connection is created or modified, or whenever the *ConnectDescTemplate* used by a connection is modified. You can enable this event with the *BeginNodeConnChangeEvent* method. You can disable this event with the *EndNodeConnChangeEvent* method. Once you have enabled this event, it will be fired whenever an application invokes the *Connect* method to create or modify a connection, or whenever an application invokes the *Disconnect* method to remove a connection. It will also be fired whenever the *ConnectDescTemplate* used by a connection is modified. When adding a new connection or removing a connection, this event will be genereated once for the connection's hub, and once for each target that has been added to the connection. When adding or removing a subset of targets from a connection, this event will be generated once for each of the affected targets. When this event is generated for a connection hub, the value of the *TargetDeviceHandle* element will match the value of the *hubDeviceHandle* element, and the value of the *TargetNvMtIndex* element will match the value of the *HubNvMtIndex* element.
Syntax	*OnNodeConnChangeEvent*(*NetworkHandle* as Long, *System Handle* as Long, *EventTag* as Integer, *TargetDeviceHandle* as Long, *Version* as Integer, *ObjectChangeType* as Integer, *TargetNvMtIndex* as Integer, *HubDeviceHandle* as Long, *HubNvMtIndex* as Long) **Element** — **Description** *NetworkHandle* — The *Handle* of the *Network* object containing the modified connection. *System Handle* — The *Handle* of the *System* object containing the modified connection. *EventTag* — Obsolete.

	TargetDeviceHandle	The *Handle* of the *AppDevice* object whose connection changed.
	Version	The version of the device's connection information. This is incremented by 1 whenever the device is added to or removed from a connection.
	ObjectChangeType	Indicates how the connection was modified. The values that can be returned in this element, which are stored in the *ConstNodeConnChangeTypes* constant, are as follows: **0 lcaConnectionTargetAdded** A new target has been added to the connection's hub *NetworkVariable* or *MessageTag*. This value maps to the **lcaObjectCreate (0)** value of the *ConstObjectChangeTypes* constant, which pre-LNS 3.20 applications used for the *OnNodeConnChangeEvent* event. **1 lcaConnectionTargetRemoved** A target has been removed from the connection's hub *NetworkVariable* or *MessageTag*. This value maps to the **lcaObjectDelete (1)** value of the *ConstObjectChangeTypes* constant, which pre-LNS 3.20 applications used for the *OnNodeConnChangeEvent* event. **2 lcaConnectionDescChanged** The *ConnectDescTemplate* used by the connection's hub *NetworkVariable* or *MessageTag* was modified.
	TargetNvMtIndex	The *Index* of the *MessageTag* or *NetworkVariable* object that was added to or removed from a connection, or the index of the connection hub whose *DeviceTemplate* was modified.
	HubDeviceHandle	The *Handle* property of the connection's hub *NetworkVariable*

		or *MessageTag*.
	HubNvMtIndex	The *Index* of the connection's hub *NetworkVariable* or *MessageTag*.
Added to API	LNS Release 3.0.	

OnNodeIntfChangeEvent

Summary	This event is generated whenever a device's external interface is changed. You can enable this event with the *BeginNodeIntfChangeEvent* method. You can disable this event with the *EndNodeIntfChangeEvent* method. The *ObjectChangeType* element contains the descriptions of the information will be contained in the *Name* and *ObjectIndex* elements returned by the event for each *ObjectChangeType* value. In addition, the value of the *ObjectChangeType* element determines when the event will be fired. **Note:** Some interface changes will cause this event to be fired as soon as the change is made in the OpenLNS database. In other cases, the event will not be fired until the physical device on the network is updated with the change. The timing of this depends on the *system management mode*.
Syntax	**OnNodeIntfChangeEvent**(*NetworkHandle* as Long, *System Handle* as Long, *EventTag* as Integer, *DeviceHandle* as Long, *Version* as Integer, *ObjectChangeType* as Integer, *ObjectIndex* as Integer, *Name* as String) **Element** — **Description** *NetworkHandle* — The *Handle* of the *Network* object containing the modified connection. *System Handle* — The *Handle* of the *System* object containing the modified connection. *EventTag* — Obsolete. *DeviceHandle* — The *Handle* of the *AppDevice* object whose connection changed. *Version* — The version of the device's interface information as a signed 16-bit value. This is incremented by one whenever the device's external interface changes. This is a signed value; therefore, the version may be reported as a positive or negative number, following this pattern: 0, 1, 2, 3….32766, 32767, -32768, -32767, -32766…-1, 0, 1, 2….32766, etc.

		You can also determine the version of an *Interface* object by reading the *Version* property.
	ObjectChangeType	Indicates how the interface changed. Changes that may be reported by the event include the addition, removal, or modification of a *NetworkVariable*, *MessageTag*, or *LonMarkObject* object on the interface. The possible values for this element, which are contained in the *ConstNodeIntfChangeTypes* constant, are as follows: **0 lcaNodeInterfaceNvAdded** A *NetworkVariable* was added to the interface. The *Name* element will contain the added network variable's *ProgrammaticName*. The *ObjectIndex* element will contain the added network variable's device index number. **1 lcaNodeInterfaceNvRemoved** A network variable was removed from the interface. The *Name* element will contain an empty string. The *ObjectIndex* element will contain the removed network variable's device index number. **2 lcaNodeInterfaceNvModified** The type or programmatic name of a network variable on the interface was modified. The *Name* element will contain an empty string. The *ObjectIndex* element will contain the modified network variable's device index number. **3 lcaNodeInterfaceNvRenamed** The user name of a network variable on the interface was modified. The *Name* element will contain the

		affected network variable's new user name. The user name is stored in the network variable's *Name* property. The *ObjectIndex* element will contain the renamed network variable's device index number. **4** **lcaNodeInterfaceLonMarkObjectAdded** A *LonMarkObject* obect (dynamic function block) was added to the interface. The *Name* element will contain the added *LonMarkObject*'s *ProgrammaticName*. The *ObjectIndex* element will contain the new *LonMarkObject* object's device index number. **5** **lcaNodeInterfaceLonMarkObjectRemoved** A *LonMarkObject* (dynamic function block) was removed from the interface. The *Name* element will contain an empty string. The *ObjectIndex* element will contain the removed *LonMarkObject* object's device index number. **6** **lcaNodeInterfaceLonMarkObjectNvMemberAssigned** A network variable was assigned to a *LonMarkObject* on the interface. The *Name* element will contain an empty string. The *ObjectIndex* element will contain the device index number of the network variable that has been added to the *LonMarkObject*. **7** **lcaNodeInterfaceLonMarkObjectNvMemberRemoved** A network variable was removed from a *LonMarkObject* on the

		interface. The *Name* element will contain an empty string. The *ObjectIndex* element will contain the affected *LonMarkObject* object's device index number. **8 lcaNodeInterfaceLonMarkObjectRenamed** The user name of a *LonMarkObject* on the interface was changed. The *Name* element will contain the *LonMarkObject*'s new user name. The user name is stored in the *LonMarkObject*'s *Name* property. The *ObjectIndex* element will contain the renamed *LonMarkObject* object's device index number. **9 lcaNodeInterfaceLonMarkObjectRenamedProgrammaticName** The programmatic name of a *LonMarkObject* on the interface was changed. The *Name* element will contain the *LonMarkObject*'s new *ProgrammaticName*. The *ObjectIndex* element will contain the renamed *LonMarkObject* object's device index number. **10 lcaNodeInterfaceMessageTagAdded** A *MessageTag* was added to the interface. The *Name* element will contain the new message tag's user name. The user name is stored in the message tag's *Name* property. The *ObjectIndex* element will contain the new *MessageTag* object's device index number. **11 lcaNodeInterfaceMessageTagRe**

		moved A message tag was removed from the interface. The *Name* element will contain an empty string. The *ObjectIndex* element will contain the removed *MessageTag* object's device index number. **12 lcaNodeInterface MessageTagRenamed** Reserved for future use. **13 lcaNodeInterface CpRenamed** The user name of a configuration property on the interface was modified. The *Name* element will contain the affected configuration property's new user name. The user name is stored in the configuration property's *Name* property. The *ObjectIndex* element will contain the renamed configuration property's device index number **14 lcaNodeInterface NvDescriptionChanged** Reserved for future use. **15 lcaNodeInterface** *LonMarkObject* **DescriptionChanged** Reserved for future use. **16 lcaNodeInterface MessageTag DescriptionChanged** Reserved for future use. **17 lcaNodeInterface CpDescriptionChanged** Reserved for future use.
	ObjectIndex	The *Index* of the message tag, network variable, or LonMarkObject that was modified. This represents the device index of the modified object. For more information on this, see the **Remarks** section.

	Name	The name of the object that was modified. This may be the user name or the programmatic name of the object, depending on the *ObjectChangeType* that caused the event. In some cases, the Name element will contain an empty string.
Added to API	LNS Release 3.0.	

OnNvMonitorPointErrorEvent

Summary	This event is generated whenever there is a write or poll failure on a network variable monitor point. If you are using a programming environment which supports multi-threading (such as Visual C++), you can receive update events on a separate thread by creating an object which implements the *ILcaNvMonitorPointListener* interface.
Syntax	*OnNvMonitorPointErrorEvent* (*nvMonitorPoint* As Object, *errorType* as Integer)

Element	**Description**
nvMonitorPoint	The *NvMonitorPoint* object that had an error.
errorType	The error type for the event. The possible values for this element, which are contained in the *ConstMonitorEventType* constant, are as follows: **0 lcaMonitorEventTypeNull** This value is not used. **1 lcaMonitorEventTypeQuit** This value is not used. **2 lcaMonitorEventTypeAdd** This value is not used. **3 lcaMonitorEventTypeRemove** This value is not used. **4 lcaMonitorEventTypeMsCreate** This value is not used. **5 lcaMonitorEventTypeMsDelete** This value is not used. **6 lcaMonitorEventTypeMsChange** This value is not used.

		7 lcaMonitorEventTypeMsError This value is not used. **8 lcaMonitorEventTypeNvCreate** A network variable monitor point has been created. **9 lcaMonitorEventTypeNvDelete** A network variable monitor point has been removed. **10 lcaMonitorEventTypeNvChange** This value is not used. **11 lcaMonitorEventTypeNvError** A network variable monitor point has returned an error. See the *OnNvMonitorPointErrorEvent* event. **12 lcaMonitorEventTypeNvUpdate** A network variable monitor point has received an update. See the *OnNvMonitorPointUpdateEvent* event. **13 lcaMonitorEventTypeNvComplete** A completion code has returned for the monitor point. **14 lcaMonitorEventTypeMsgCreate** A message monitor point has been created. **15 lcaMonitorEventTypeMsgDelete** A message monitor point has been removed. **16 lcaMonitorEventTypeMsgChange** This value is not used. **17 lcaMonitorEventTypeMsgError** A message monitor point has returned an error. See the *OnMsgMonitorPointErrorEvent* event. **18 lcaMonitorEventTypeMsgUpdate** A message monitor point has received an update. See the *OnMsgMonitorPointUpdateEvent* event.

		19 lcaMonitorEventTypeMsgRequest A message monitor point has received a request message. **20 lcaMonitorEventTypeMsgResponse** A message monitor point has received a reponse message. **21 lcaMonitorEventTypeMsgComplete** This value is not used.
Added to API	LNS Release 3.0.	

OnNvMonitorPointEvent

Summary	This event is generated whenever a network variable monitor point is created, removed, enabled or disabled. This event is fired whenever a network variable monitor point is *created* or *removed* from a permanent monitor set. The events are only fired in this case if the monitor set is open. In addition, when you *close* or *open* a permanent monitor set, this event will be fired for each network variable monitor point in the set, indicating that the monitor points have been removed from or created in memory. This event is also fired the first time you *enable* a network variable monitor point in a temporary monitor set after creating the temporary monitor set. It will also be fired when you remove a network variable monitor point from a temporary monitor set, as long as that monitor point had been enabled.
Syntax	*OnNvMonitorPointEvent*(*nvMonitorPoint* As Object, *EventType* as Integer) **Element** — **Description** *nvMonitorPoint* — The *NvMonitorPoint* object that was created or shut down. *EventType* — The event code for the event. The possible values for this element, which are contained in the *ConstMonitorEventType* constant, are as follows: **0 lcaMonitorEventTypeNull** This value is not used. **1 lcaMonitorEventTypeQuit** This value is not used. **2 lcaMonitorEventTypeAdd**

		This value is not used. **3 lcaMonitorEventTypeRemove** This value is not used. **4 lcaMonitorEventTypeMsCreate** This value is not used. **5 lcaMonitorEventTypeMsDelete** This value is not used. **6 lcaMonitorEventTypeMsChange** This value is not used. **7 lcaMonitorEventTypeMsError** This value is not used. **8 lcaMonitorEventTypeNvCreate** A network variable monitor point has been created. **9 lcaMonitorEventTypeNvDelete** A network variable monitor point has been removed. **10 lcaMonitorEventTypeNvChange** This value is not used. **11 lcaMonitorEventTypeNvError** A network variable monitor point has returned an error. See the *OnNvMonitorPointErrorEvent* event. **12 lcaMonitorEventTypeNvUpdate** A network variable monitor point has received an update. See the *OnNvMonitorPointUpdateEvent* event. **13 lcaMonitorEventTypeNvComplete** A completion code has returned for the monitor point. **14 lcaMonitorEventTypeMsgCreate** A message monitor point has been created. **15 lcaMonitorEventTypeMsgDelete** A message monitor point has been removed. **16 lcaMonitorEventTypeMsgChange**

		This value is not used. **17 lcaMonitorEventTypeMsgError** A message monitor point has returned an error. See the *OnMsgMonitorPointErrorEvent* event. **18 lcaMonitorEventTypeMsgUpdate** A message monitor point has received an update. See the *OnMsgMonitorPointUpdateEvent* event. **19 lcaMonitorEventTypeMsgRequest** A message monitor point has received a request message. **20 lcaMonitorEventTypeMsgResponse** A message monitor point has received a reponse message. **21 lcaMonitorEventTypeMsgComplete** This value is not used.
Added to API	LNS Release 3.0.	

OnNvMonitorPointUpdateEvent

Summary	This event is generated whenever a network variable monitor point update is received. If you are using a programming environment which supports multi-threading (such as Visual C++), you can receive update events on a separate thread by creating an object which implements the *ILcaMsgMonitorPointListener* interface.
Syntax	*OnNvMonitorPointUpdateEvent* (*nvMonitorPoint* As Object, *DataPoint* as Object, *Src* as Object) **Element** — **Description** *nvMonitorPoint* — The *NvMonitorPoint* object that received an update. *DataPoint* — A *DataPoint* object containing the received value. The *DataPoint* parameter's *AutoRead* property is set to TRUE. *Src* — A *SourceAddress* object indicating the source device of the update.
Added to API	LNS Release 3.0.

OnNVUpdateError

Summary	This event is generated whenever a monitored network variable update fails. This error occurs when the Data Server gets a network error when it attempts the get the value of a network variable which is being monitored.
Syntax	**OnNvUpdateError**(*monitorTag* As Long, *dsError* as Long) **Element** — **Description** *monitorTag* — The monitor tag of the network variable which had an update failure. *dsError* — The type of update failure. This element will always have a value of 200 (network error).
Added to API	LNS Release 3.0.

OnSessionChangeEvent

Summary	This event is generated whenever the state of an xDriver session changes, or whenever the state of your client's *NetworkServiceDevice* changes. This event will then be fired each time the state of an xDriver session your client application is handling changes, including when the session is disconnected for any reason.

<table>
<tr><td></td><td>This event is also fired each time the state of your client's NetworkServiceDevice changes from the online state to the offline state, or vice versa. You could use this event to keep track of when the NetworkServiceDevice is offline, as polling is suspended, and monitor and control events will not be delivered to your application, while the NetworkServiceDevice is in the offline state. You can also check the state of the NetworkServiceDevice by reading the State property of the AppDevice object that represents the NetworkServiceDevice.</td></tr>
<tr><td>Syntax</td><td>OnSessionChangeEvent(networkName, sessionState)

Element — Description

networkName — A String containing the name of the network involved in the session.

sessionState — The current status of the xDriver session, or of the Network Service Device.

The values that can be returned as this element are as follows (these values are stored in the ConstSessionStates constant):

0 lcaSessionStateClosed

The xDriver session has been closed.

1 lcaSessionStateConnecting

The connection to the RNI device involved in the session has been lost, and xDriver is attempting to reconnect to the RNI.

If xDriver is able to reestablish the connection, the event will be fired again with the lcaSessionStateEstablished (2) value as the sessionState element.

If xDriver is not able to reestablish the connection, the event will be fired again with the lcaSessionStateFailed (3) value as the sessionState element.

2 lcaSessionStateEstablished

xDriver has successfully reestablished connection to the RNI device in the connection, after communication with the device was lost.

3 lcaSessionStateFailed

Connection between your application and the RNI device has</td></tr>
</table>

	been lost, and recovery is either disabled, or it has timed out. Your application will no longer be able to communicate with the remote network. **4 lcaSessionStateOffline** The *NetworkServiceDevice* has been taken offline, possibly because the device is being commissioned. Polling is suspended and monitor and control events will not be delivered to your application while the *NetworkServiceDevice* is offline. **5 lcaSessionStateOnline** The *NetworkServiceDevice* has returned to the online state.
Added to API	LNS Release 3.20.

OnSystemMgmtModeChangeEvent

Summary	Once an application registers for this event, it is generated each time the system management mode changes. The current system management mode for a system is stored in the *MgmtMode* property of the *System* object. You can register your application for this event with the *BeginSystemMgmtModeChangeEvent* method. You can disable the event with the *EndSystemMgmtModeChangeEvent* method. You can read or write to the system management mode with the *MgmtMode* property of the *System* object.
Syntax	**OnSystemMgmtModeChangeEvent**(*networkHandle, systemHandle, mgmtMode)* **Element** — **Description** *networkHandle* — The handle of the *Network* where the system management mode changed. *systemHandle* — The handle of the *System* where the management mode changed. *mgmtMode* — The current system management mode. The values that can be returned for this element, which are contained in the *ConstMgmtModes* constant, are as follows: **0 lcaMgmtModePropagateConfigUpdates** Network configuration changes are

	applied to both the OpenLNS database and the physical devices. **1 lcaMgmtModeDeferConfigUpdates** Network configuration changes are applied only to the OpenLNS network database.
Added to API	LNS Release 3.20.

OnSystemNssIdle

Summary	Indicates that an OpenLNS Server idle message was raised. When you call the *BeginNssIdleEvent* method to enable this event, you will specify an interval. This event will then be fired at that interval while your application is waiting for lengthy network operations to complete. Instances of this event will be returned synchronously, and if your application does not handle the event in a timely manner, then your application may hang. This event allows your application to execute code while OpenLNS is busy with an operation. This may be the case when you change the value of the *MgmtMode* property from **lcaMgmtModeDeferConfigUpdates (1)** to **lcaMgmtModePropagateConfigUpdates (0)**, or if you are commissioning a device. The main thread of your application will need to wait for these operations to complete, so you could use this event to refresh the display of your client application, so that the user knows it is not stuck. You can only make the following OpenLNS calls from within the this event's handler: • You can access the *ServiceStatus* property to determine the status of the service OpenLNS is trying to perform. • If an operation is taking too long for OpenLNS to execute, you can cancel it with the *CancelTransaction* method from the event handler. • If you are performing a network recovery, you can access the *RecoveryStatus* property from the event handler, and then determine the status of the network recovery. You can disable this event with the *EndNssIdleEvent* method.

Syntax	**OnSystemNssIdle**()
Added to API	Prior to LNS Release 3.0.

OnSystemServicePin

Summary	Indicates that a qualifying service pin message has been received. You must call the *System* object's *BeginServicePinEvent* method to enable service pin events. The a*ppDeviceObject* parameter contains the application device that originated the service pin event. The *eventTag* parameter contains the corresponding event tag. You can call the *EndServicePinEvent* method to disable the service pin events. If the service pin was pressed on a router, the *auxClassId* parameter will contain **lcaClassIdRouter**; otherwise it will contain **lcaClassIdObjectServer**.
Syntax	**OnSystemServicePin**(*networkHandle* as Long, *systemHandle* as Long, *eventTag* as Integer, *neuronId* as String, *programId* as String, *location* as String, *channelHandle* as Long, *auxClassId* as Integer, *objectHandle* as Long)

Element	Description
networkHandle	The *Handle* of the *Network* object containing the modified connection.
systemHandle	The *Handle* of the *System* object containing the modified connection.
eventTag	The Event tag value corresponding to this event, set by the *BeginServicePinEvent* method.
neuronId	The Neuron ID returned by the service pin event.
programId	The Program ID returned by the service pin event.
location	The location string of the device that caused the event.
channelHandle	The handle of the channel in which the service pin event occured.
auxClassId	Indicates the type of device whose service pin was pressed.
objectHandle	The handle of the object whose service pin was grounded. If the device that caused the event has not been registered, this will return **0**.

Added to API	Prior to LNS Release 3.0.

Validate

Summary	This is a standard ActiveX control event used for visual controls. This event is not applicable for the Object Server.

ObjectStatus

An *ObjectStatus* object can be accessed through the *Status* property or the *ReportMask* property of a *LonMarkObject*.

If the object was accessed through the *Status* property, the object represents a status report. In this case, the properties of the *ObjectStatus* object reflect the current status of the *LonMarkObject*. Some LonMarkObjects, however, do not support all of the status attributes contained in an *ObjectStatus* object; therefore, some of the information contained in this object may not be useful.

You can use the *ReportMask* property to determine which status attributes are supported. If you access an *ObjectStatus* object through the *ReportMask* property, the object represents a report mask. In this case, the values of each property of the *ObjectStatus* object reflect whether the *LonMarkObject* supports the related status attribute.

For example, consider the *ResetComplete* property of the *ObjectStatus* object. The *ResetComplete* property contained in an *ObjectStatus* object accessed through the *Status* property indicates whether the most recent reset of the *LonMarkObject* has completed. The *ResetComplete* property contained in an *ObjectStatus* object accessed through the *ReportMark* property indicates whether or not the *LonMarkObject* can be reset. If the *LonMarkObject* cannot be reset, then the *ResetComplete* property in the *ObjectStatus* object accessed through the *Status* property would not contain useful information.

Note: Some properties contain the same information, whether the *LonMarkObject* represents a report mask or a status report. These properties include the *ClassId*, *InvalidId*, *InvalidRequest*, *ObjectId*, *Parent*, and the *Summary* properties.

The device containing the *LonMarkObject* must be in the configured/online state (**lcaStateCnfgOnline**) to successfully read the properties in this object. The *LonMarkObject's Request* property can be used to refresh or clear the information contained in the *Status* and *ReportMask* properties. See the *LonMark Application Layer Interoperability Guidelines* for more information on LonMarkObjects.

The following table summarizes the *ObjectStatus* object.

Description	A status report for a *LonMarkObject* or a report of applicable status attributes.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*LonMarkObject* object.
Default Property	None.
Methods	None.
Properties	• *AlarmNotifyDisabled* • *ClassId* • *CommFailure* • *Disabled* • *ElectricalFault* • *FailSelfTest* • *FeedbackFailure*

	• *InAlarm* • *InOverride* • *InvalidId* • *InvalidRequest* • *LockedOut* • *ManualControl* • *MechanicalFault* • *ObjectId* • *OpenCircuit* • *OutOfLimits* • *OutOfService* • *OverRange* • *Parent* • *ProgrammingFail* • *ProgrammingMode* • *ReportMask* • *ResetComplete* • *SelfTestInProgress* • *Summary* • *UnableToMeasure* • *UnderRange*

Methods

The *ObjectStatus* object does contain any methods.

Properties

The *ObjectStatus* object contains the following properties.

- *AlarmNotifyDisabled*
- *ClassId*
- *CommFailure*
- *Disabled*
- *ElectricalFault*
- *FailSelfTest*
- *FeedbackFailure*
- *InAlarm*
- *InOverride*
- *InvalidId*
- *InvalidRequest*
- *LockedOut*
- *ManualControl*
- *MechanicalFault*
- *ObjectId*
- *OpenCircuit*
- *OutOfLimits*
- *OutOfService*
- *OverRange*
- *Parent*
- *ProgrammingFail*
- *ProgrammingMode*
- *ReportMask*

- *ResetComplete*
- *SelfTestInProgress*
- *Summary*
- *UnableToMeasure*
- *UnderRange*

AlarmNotifyDisabled

Summary	Indicates whether alarm notification is disabled on the *LonMarkObject*, or whether alarm notification can be disabled on the *LonMarkObject*. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether alarm notification on the *LonMarkObject* is currently disabled. If this property is True, alarm notification has been disabled. • *ReportMask* property. Indicates whether alarm notification can be disabled on the *LonMarkObject*. If this property is True, then alarm notification can be disabled. If alarm notification on the *LonMarkObject* can be disabled, you can disable it by writing to the **lcaLonMarkObjectRequestAlarmNotifyDisabled (12)** value to the *Request* property of the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*notifyDisabledFlag*= *statusObject*.**AlarmNotifyDisabled** **Element** — **Description** *notifyDisabledFlag* — A Boolean value indicating whether alarm notification on the *LonMarkObject* has been disabled, or whether alarm notification on the *LonMarkObject* can be disabled. **Status Property** **TRUE**. Alarm notification on the *LonMarkObject* has been disabled. **FALSE**. Alarm notification has not been disabled. **ReportMask Property** **TRUE**. Alarm notification on the *LonMarkObject* can be disabled. **FALSE**. Alarm notification can not be disabled. *statusObject* — The *ObjectStatus* object to be acted

	on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *ObjectStatus* object in the *ConstClassIds* constant: **46 lcaClassIdObjectStatus** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CommFailure

Summary	Indicates whether there has been a communication failure on the *LonMarkObject* object, or whether the *LonMarkObject* could report such a failure. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether a communication failure on the *LonMarkObject* has occurred. If this property is True, then there has been a communication failure. • *ReportMask* property. Indicates whether the *LonMarkObject* can report a communications failure through the *ObjectStatus* object when it is accessed as a status report. If this property is True, then it can report a failure.
Availability	Local, full, and lightweight clients.
Syntax	*commFailFlag* = *statusObject*.**CommFailure** **Element** — **Description** *commFailFlag* — A Boolean value indicating whether a communication failure on the *LonMarkObject* has occurred, or whether the *LonMarkObject* can report a communications failure. **Status Property** **TRUE**. A communication failure on the *LonMarkObject* has occurred. **FALSE**. A communication failure has not occurred. **ReportMask Property** **TRUE**. The *LonMarkObject* can report a communications failure. **FALSE**. The *LonMarkObject* can not report a communications failure. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Disabled

Summary	Indicates whether the *LonMarkObject* is currently disabled, or whether the *LonMarkObject* can be disabled. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is currently disabled. If this property is True, then the *LonMarkObject* is currently disabled. • *ReportMask* property. Indicates whether the *LonMarkObject* can be disabled. If this property is True, then the *LonMarkObject* can be disabled. If the *LonMarkObject* can be disabled, you can do so by writing the **lcaLonMarkObjectRequestDisabled (1)** value to the *Request* property of the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*disableFlag* = *statusObject*.**Disabled** **Element** — **Description** *disableFlag* — A Boolean value indicating whether whether the *LonMarkObject* is currently disabled, or whether the *LonMarkObject* can be disabled. **Status Property** **TRUE**. The *LonMarkObject* is disabled. **FALSE**. The *LonMarkObject* is not disabled. **ReportMask Property** **TRUE**. The *LonMarkObject* can be disabled. **FALSE**. The *LonMarkObject* can not be disabled. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ElectricalFault

Summary	Indicates whether an electrical fault has been detected in the *LonMarkObject* object, or whether the *LonMarkObject* could report an electrical fault. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether an electrical fault has been detected in the *LonMarkObject*. If this property is True, then an electrical fault has been detected in the *LonMarkObject*. • *ReportMask* property. Indicates whether the *LonMarkObject* could report an electrical fault. If this property is True, then the *LonMarkObject* could report an electrical fault.
Availability	Local, full, and lightweight clients.
Syntax	*eFaultFlag* = *statusObject*. **ElectricFault** **Element** — **Description** *eFaultFlag* — A Boolean value indicating whether an electrical fault has been detected in the *LonMarkObject* object, or whether the *LonMarkObject* could report an electrical fault. **Status Property** **TRUE**. An electrical fault has been detected in the *LonMarkObject* object. **FALSE**. An electrical fault has not been detected. *ReportMask* **Property** **TRUE**. The *LonMarkObject* can report an electrical fault. **FALSE**. The *LonMarkObject* can not report an electrical fault. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FailSelfTest

Summary	Indicates whether the *LonMarkObject* passed its most recent self-test, or whether the *LonMarkObject* can perform a self-test. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* failed its most recent self-test. If this property is True, then the *LonMarkObject* failed the self-test. • ReportMask property. Indicates whether the *LonMarkObject* can perform a self-test.
Availability	Local, full, and lightweight clients.
Syntax	*failTestFlag* = *statusObject*. **FailSelfTest** **Element** — **Description** *eFaultFlag* — A Boolean value indicating whether the *LonMarkObject* passed its most recent self-test, or whether the *LonMarkObject* can perform a self-test. **Status Property** **TRUE**. The *LonMarkObject* failed its most recent self-test. **FALSE**. The *LonMarkObject* passed its most recent self-test. **ReportStatus Property** **TRUE**. The *LonMarkObject* can perform a self-test. **FALSE**. The *LonMarkObject* can not perform a self-test. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FeedbackFailure

Summary	Indicates whether the feedback signal of the *LonMarkObject* is being received, or whether the *LonMarkObject* is capable of reporting this information. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the feedback signal of the *LonMarkObject* is being received. • *ReportMask* property. Indicates whether the *LonMarkObject* can report a feedback failure through the *ObjectStatus* object when it is is accessed as a status report.
Availability	Local, full, and lightweight clients.
Syntax	*failTestFlag* = *statusObject*. **FailSelfTest** **Element** — **Description** *eFaultFlag* — A Boolean value indicating whether the feedback signal of the *LonMarkObject* is being received, or whether the *LonMarkObject* is capable of reporting this information. **Status Property** **TRUE**. The feedback signal of the *LonMarkObject* is being received. **FALSE**. The feedback signal of the *LonMarkObject* is not being received. **ReportMask Property** **TRUE**. The *LonMarkObject* can report a feedback failure through the *ObjectStatus* object. **FALSE**. The *LonMarkObject* cannot report a feedback failure through the *ObjectStatus* object. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

InAlarm

Summary	Indicates whether the *LonMarkObject* is currently in an alarm condition, or whether the *LonMarkObject* is capable of reporting an alarm condition. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is an alarm condition. If this property is True, then the *LonMarkObject* is an alarm condition. • *ReportMask* property. Indicates whether the *LonMarkObject* can report an an alarm condition. If this property is True, then the *LonMarkObject* can report an alarm condition. You can get more information about the alarm status of a *LonMarkObject* by reading its *LonMarkAlarm* property.
Availability	Local, full, and lightweight clients.
Syntax	*inAlarmFlag = statusObject.* **InAlarm** **Element** — **Description** *inAlarmFlag* — A Boolean value indicating whether whether the *LonMarkObject* is currently in an alarm condition, or whether the *LonMarkObject* can report an alarm condition. **Status Property** **TRUE**. The *LonMarkObject* is in an alarm condition. **FALSE**. The *LonMarkObject* is not in an alarm condition. **ReportMask Property** **TRUE**. The *LonMarkObject* can report an alarm condition disabled. **FALSE**. The *LonMarkObject* can not report an alarm condition. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

InOverride

Summary	Indicates whether the *LonMarkObject* is currently in an override state, or whether the *LonMarkObject* can be overridden. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is an override state. If this property is True, then the *LonMarkObject* is an override state. • *ReportMask* property. Indicates whether the *LonMarkObject* can be overridden. If this property is True, then the *LonMarkObject* can be overridden. If the *LonMarkObject* can be overridden, you can initiate an override by writing the **lcaLonMarkObjectRequestOverride (6)** value to the *Request* property of the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*overrideFlag* = *statusObject*. **InOverride** **Element** — **Description** *overrideFlag* — A Boolean value indicating whether the *LonMarkObject* is currently in an override state, or whether the *LonMarkObject* can be overridden. **Status Property** **TRUE**. The *LonMarkObject* is in an override state. **FALSE**. The *LonMarkObject* is not in an override state. **ReportMask Property** **TRUE**. The *LonMarkObject* can be overridden. **FALSE**. The *LonMarkObject* can not be overridden. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

InvalidId

Summary	Indicates that the requested *LonMarkObject* does not exist.
Availability	Local, full, and lightweight clients.
Syntax	*invalidIdFlag* = *statusObject*.**InvalidId** **Element** — **Description** *invalidIdFlag* — A Boolean value indicating whether the *LonMarkObject* exists. **TRUE**. The ID used to reference the *LonMarkObject* is invalid; therefore, the *LonMarkObject* does not exist. **FALSE**. The *LonMarkObject* exists. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

InvalidRequest

Summary	Indicates whether a request that is not supported by this *LonMarkObject* object has been made. You can send a request to a *LonMarkObject* by writing to its *Request* property. See the *LonMarkObject's Request* property for more information.
Availability	Local, full, and lightweight clients.
Syntax	*invalReqFlag* = *statusObject*.**InvalidRequest** **Element** — **Description** *invalReqFlag* — A Boolean value indicating whether a request that is not supported by this *LonMarkObject* object has been made. **TRUE**. An invalid request has been made. You can determine which set of requests a *LonMarkObject* supports by reading the other properties of the *ObjectStatus* object returned through the *LonMarkObject's ReportMask* property. **FALSE**. An invalid request has not

	been made. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LockedOut

Summary	Indicates whether the *LonMarkObject* is in an operable state, or whether the *LonMarkObject* can report this information. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is an operable state. If this property is True, then the *LonMarkObject* is online, but is not in an operable state. • *ReportMask* property. Indicates whether the *LonMarkObject* can report that it is locked out and inoperable.
Availability	Local, full, and lightweight clients.
Syntax	*lockedOutFlag* = *statusObject*. **LockedOut** **Element** — **Description** *lockedOutFlag* — A Boolean value indicating whether the *LonMarkObject* is in an operable state, or whether the *LonMarkObject* can report that it is locked out and inoperable. **Status Property** **TRUE**. The *LonMarkObject* is online, but is not in an operable state. **FALSE**. The *LonMarkObject* is online, and it is an operable state. **ReportMask Property** **TRUE**. The *LonMarkObject* can report that it is locked out and inoperable. **FALSE**. The *LonMarkObject* can not report that it is locked out and inoperable. *statusObject* — The *ObjectStatus* object to be acted on.

Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ManualControl

Summary	Indicates whether the *LonMarkObject* is under manual control, or whether the *LonMarkObject* can be placed under manual control. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is under manual control. If this property is True, then the *LonMarkObject* is under manual control. • *ReportMask* property. Indicates whether the *LonMarkObject* can be placed under manual control.
Availability	Local, full, and lightweight clients.
Syntax	*manControlFlag* = *statusObject*. **ManualControl** **Element** — **Description** *manControlFlag* — A Boolean value indicating whether the *LonMarkObject* is in under manual control, or whether the *LonMarkObject* can be manually controlled. **Status Property** **TRUE**. The *LonMarkObject* is under manual control. **FALSE**. The *LonMarkObject* is not under manual control. **ReportMask Property** **TRUE**. The *LonMarkObject* can be manually controlled. **FALSE**. The *LonMarkObject* can not be manually controlled. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

MechanicalFault

Summary	Indicates whether a mechanical fault has been detected in the *LonMarkObject* object, or whether the *LonMarkObject* could report a mechanical fault. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether a mechanical fault has been detected in the *LonMarkObject*. If this property is True, then a mechanical fault has been detected in the *LonMarkObject*. • *ReportMask* property. Indicates whether the *LonMarkObject* could report a mechanical fault. If this property is True, then the *LonMarkObject* could report a mechanical fault.
Availability	Local, full, and lightweight clients.
Syntax	*mechFaultFlag* = *statusObject*. **MechanicalFault** **Element** — **Description** *mechFaultFlag* — A Boolean value indicating whether a mechanical fault has been detected in the *LonMarkObject* object, or whether the *LonMarkObject* could report a mechanical fault. **Status Property** **TRUE**. A mechanical fault has been detected in the *LonMarkObject* object. **FALSE**. A mechanical fault has not been detected. **ReportMask Property** **TRUE**. The *LonMarkObject* can report a mechanical fault. **FALSE**. The *LonMarkObject* can not report a mechanical fault. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ObjectId

Summary	Contains the ID of the *LonMarkObject* to which this *ObjectStatus* object applies. Objects are numbered from 0 to n-1, where *n* is the number of objects in an *AppDevice*. See the LonMark Application Layer Interoperability Guidelines for more information.
Availability	Local, full, and lightweight clients.
Syntax	*objectIdValue* = *statusObject*.**ObjectId** **Element** — **Description** *objectIdValue* — The ID of the *LonMarkObject* to which this *ObjectStatus* object applies *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

OpenCircuit

Summary	Indicates whether an open circuit has been discovered in the *LonMarkObject* object, or whether the *LonMarkObject* could report an open circuit. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether an open circuit has been discovered in the *LonMarkObject* object. If this property is True, then open circuit has been discovered. • *ReportMask* property. Indicates whether the *LonMarkObject* could report that an open circuit has been discovered. If this property is True, then the *LonMarkObject* could report an open circuit.
Availability	Local, full, and lightweight clients.
Syntax	*openCircuitFlag* = *statusObject*.**OpenCircuit** **Element** — **Description** *openCircuitFlag* — A Boolean value indicating whether an open circuit has been discovered in the *LonMarkObject* object, or whether the *LonMarkObject* could report an open circuit. **Status Property** **TRUE**. An open circuit has been discovered in the

	LonMarkObject object. **FALSE**. An open circuit has not been discovered. **ReportMask Property** **TRUE**. The *LonMarkObject* can report an open circuit. **FALSE**. The *LonMarkObject* can not report an open circuit. *statusObject* The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

OutOfLimits

Summary	Indicates whether the *LonMarkObject* has exceeded its alarm limits, or whether the *LonMarkObject* is capable of reporting this information. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* has exceeded its alarm limits. If this property is True, then the alarm limits have been exceeded. • *ReportMask* property. Indicates whether the *LonMarkObject* could report that it has exceeded its alarm limits. If this property is True, then the *LonMarkObject* could report such information.
Availability	Local, full, and lightweight clients.
Syntax	*limitsFlag* = *statusObject*.**OutOfLimits** **Element** **Description** *limitsFlag* A Boolean value indicating whether the *LonMarkObject* has exceeded its alarm limits, or whether the *LonMarkObject* could report that it has exceeded its alarm limits. **Status Property** **TRUE**. The *LonMarkObject* has exceeded its alarm limits. **FALSE**. The *LonMarkObject* has not exceeded its alarm limits. **ReportMaskProperty**

	TRUE. The *LonMarkObject* can report that it has exceeded its alarm limits. **FALSE**. The *LonMarkObject* can not report that it has exceeded its alarm limits. *statusObject* The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

OutOfService

Summary	Indicates whether the *LonMarkObject* is currently operational, or whether the *LonMarkObject* is capable of reporting this information. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is currently operational. If this property is True, then the *LonMarkObject* is currently inoperable. • *ReportMask* property. Indicates whether the *LonMarkObject* could report that is inoperable. If this property is True, then the *LonMarkObject* could report such information.
Availability	Local, full, and lightweight clients.
Syntax	*serviceFlag* = *statusObject*. **OutOfService** **Element** **Description** *serviceFlag* A Boolean value indicating whether the *LonMarkObject* is currently operational, or whether the *LonMarkObject* could report that it is operational. **Status Property** **TRUE**. The *LonMarkObject* is currently inoperable. **FALSE**. The *LonMarkObject* is currently operational. *ReportMask* **Property** **TRUE**. The *LonMarkObject* can report that it is inoperable. **FALSE**. The *LonMarkObject* can not

	report that it is inoperable. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

OverRange

Summary	Indicates whether the value associated with the *LonMarkObject* is greater than its acceptable range, or whether the *LonMarkObject* is capable of reporting this information. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the the value associated with the *LonMarkObject* is greater than its acceptable range. If this property is True, then the value is over its range. • *ReportMask* property. Indicates whether the *LonMarkObject* could report that the value associated with it is over its range. If this property is True, then the *LonMarkObject* could report such information.
Availability	Local, full, and lightweight clients.
Syntax	*rangeFlag* = *statusObject*.**OverRange** **Element** — **Description** *rangeFlag* — A Boolean value indicating whether the value associated with the *LonMarkObject* is greater than its acceptable range, or whether the *LonMarkObject* could report that its value is over the acceptable range. **Status Property** **TRUE**. The value associated with the *LonMarkObject* is greater than its acceptable range. **FALSE**. The value is within the acceptable range. **ReportMaskProperty** **TRUE**. The *LonMarkObject* can report that its value is over the acceptable range. **FALSE**. The *LonMarkObject* can not

	report that its value is over the acceptable range. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ProgrammingFail

Summary	Indicates whether there has been a programming failure on the *LonMarkObject* object, or whether the *LonMarkObject* could report such a failure. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether a programming failure on the *LonMarkObject* has occurred. If this property is True, then there has been a programming failure. • *ReportMask* property. Indicates whether the *LonMarkObject* can report a programmings failure through the *ObjectStatus* object when it is accessed as a status report. If this property is True, then it can report a

	failure.
Availability	Local, full, and lightweight clients.
Syntax	*progFailFlag= statusObject.***ProgrammingFail** **Element** — **Description** *progFailFlag* — A Boolean value indicating whether a programming failure on the *LonMarkObject* has occurred, or whether the *LonMarkObject* can report a programmings failure. **Status Property** **TRUE**. A programming failure on the *LonMarkObject* has occurred. **FALSE**. A programming failure has not occurred. **ReportMaskProperty** **TRUE**. The *LonMarkObject* can report a programmings failure. **FALSE**. The *LonMarkObject* can not report a programmings failure. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ProgrammingMode

Summary	Indicates whether the *LonMarkObject* is currently in programming mode, or whether the *LonMarkObject* can be placed in programming mode. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the *LonMarkObject* is programming mode. If this property is True, then the *LonMarkObject* is programming mode. • ReportMask property. Indicates whether the *LonMarkObject* can be overridden. If this property is True, then the *LonMarkObject* can be overridden.
Availability	Local, full, and lightweight clients.

Syntax	*progModeFlag= statusObject.***ProgrammingMode** **Element** — **Description** *progModeFlag* — A Boolean value indicating whether the *LonMarkObject* is currently in programming mode, or whether the *LonMarkObject* can placed in programming mode. **Status Property** **TRUE**. The *LonMarkObject* is in programming mode. **FALSE**. The *LonMarkObject* is not in programming mode. **ReportMask Property** **TRUE**. The *LonMarkObject* can be placed in programming mode. **FALSE**. The *LonMarkObject* can not be placed in programming mode. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ReportMask

Summary	Indicates whether this *ObjectStatus* object represents a report mask, or a status report.
Availability	Local, full, and lightweight clients.
Syntax	*reportMaskFlag = objStatusObject.ReportMask* **Element** — **Description** *reportMaskFlag* — A Boolean value indicating whether the *ObjectStatus* object represents a report mask or a status report. **TRUE**. The *ObjectStatus* object represents a report mask. The *ObjectStatus* object identifies which status attributes are supported by the *LonMarkObject* object that owns this *ObjectStatus* object. These *ObjectStatus* objects

		can be accessed through the *ReportMask* property of the *LonMarkObject* **FALSE**. The *ObjectStatus* object represents a status report. The *ObjectStatus* object reflects the current status of the *LonMarkObject*. These *ObjectStatus* objects can be accessed through the *Status* property of the *LonMarkObject*.
	statusObject	The *ObjectStatus* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ResetComplete

Summary	Indicates whether the most recent reset of the *LonMarkObject* has been completed, or whether the *LonMarkObject* can be reset. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the most recent reset of the *LonMarkObject* has completed. If this property is True, then the most recent reset has been completed. • *ReportMask* property. Indicates whether the *LonMarkObject* can be reset. If this property is True, then the *LonMarkObject* can be reset. If the *LonMarkObject* can be reset, you can reset it by writing the **lcaLonMarkObjectRequestReset (17)** value to the *Request* property of the *LonMarkObject*.	
Availability	Local, full, and lightweight clients.	
Syntax	*completed = objectStatus.ResetComplete*	
	Element	**Description**
	completed	A Boolean value indicating whether the most recent reset of the *LonMarkObject* has been completed, or whether the *LonMarkObject* can be reset. **Status Property** **TRUE**. The most recent reset of the

	LonMarkObject has been completed. **FALSE**. The reset has not been completed. **ReportMask Property** **TRUE**. The *LonMarkObject* can be reset. **FALSE**. The *LonMarkObject* can not be reset. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

SelfTestInProgress

Summary	Indicates whether the most recent self-test of the *LonMarkObject* is still in progress, or whether a self-test can be performed on the *LonMarkObject*. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the most recent self-test of the *LonMarkObject* is still in progress. If this property is True, then the most recent self-test is still in progress. • *ReportMask* property. Indicates whether a self-test can be performed on the *LonMarkObject*. If this property is True, then a self-test can be performed on the *LonMarkObject*. If the *LonMarkObject* can perform a self-test, you can initiate one by writing the **lcaLonMarkObjectRequestSelfTest (3)** value to the *Request* property of the *LonMarkObject*.
Availability	Local, full, and lightweight clients.
Syntax	*selfTestFlag* = *statusObject*.**SelfTestInProgress** **Element** — **Description** *selfTestFlag* — A Boolean value indicating whether the most recent self-test of the *LonMarkObject* is still in progress, or whether a self-test can be performed on the *LonMarkObject*. **Status Property**

		TRUE. The most recent self-test of the *LonMarkObject* is still in progress. **FALSE**. The self-test has been completed. **ReportMask Property** **TRUE**. A self-test can be performed on the *LonMarkObject*. **FALSE**. A self-test can not be performed on the *LonMarkObject*.
	statusObject	The *ObjectStatus* object to be acted on.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Summary

Summary	A Long that encompasses all of the status bits in the *ObjectStatus* object. This property returns **0** if all the status bits are set to **0**. Otherwise it returns the value of all of the status bits ORed together. This provides a quick way to determine whether there are any problems without having to check every bit. You can read each bit separately by reading the other properties of the *ObjectStatus* object.	
Availability	Local, full, and lightweight clients.	
Syntax	*summaryValue* = *statusObject*.**Summary**	
	Element	**Description**
	summaryValue	The status bits from the specified *ObjectStatus* object
	statusObject	The *ObjectStatus* object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

UnableToMeasure

Summary	Indicates whether an input/output line failure has been detected in the *LonMarkObject* object, or whether the *LonMarkObject* could report an input/output line failure. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether an input/output line failure has been detected in the *LonMarkObject*. If this property is True, then an input/output line failure has been detected in the *LonMarkObject*. • *ReportMask* property. Indicates whether the *LonMarkObject* could report an input/output line failure. If this property is True, then the *LonMarkObject* could report an input/output line failure.
Availability	Local, full, and lightweight clients.
Syntax	*measureFlag* = *statusObject*.**UnableToMeasure** **Element** — **Description** *measureFlag* — A Boolean value indicating whether an input/output line failure has been detected in the *LonMarkObject* object, or whether the *LonMarkObject* could report an input/output line failure. **Status Property** **TRUE**. An input/output line failure has been detected in the *LonMarkObject* object. **FALSE**. An input/output line failure has not been detected. **ReportMask Property** **TRUE**. The *LonMarkObject* can report an input/output line failure. **FALSE**. The *LonMarkObject* can not report an input/output line failure. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

UnderRange

Summary	Indicates whether the value associated with the *LonMarkObject* is less than its acceptable range, or whether the *LonMarkObject* is capable of reporting this information. This description of this property depends on whether it was accessed through the *Status* or *ReportMask* properties of a *LonMarkObject*. • *Status* property. Indicates whether the value associated with the *LonMarkObject* is lower than its acceptable range. • *ReportMask* property. Indicates whether the *LonMarkObject* can report if it is under the range of its acceptable values through the *ObjectStatus* object.
Availability	Local, full, and lightweight clients.
Syntax	*measureFlag* = *statusObject*.**UnableToMeasure** **Element** — **Description** *measureFlag* — A Boolean value indicating whether the value associated with the *LonMarkObject* is less than its acceptable range, or whether the *LonMarkObject* is capable of reporting this information. **Status Property** **TRUE**. The value associated with the *LonMarkObject* is lower than its acceptable range. **FALSE**. The value associated with the *LonMarkObject* is not lower than its acceptable range. **ReportMask Property** **TRUE**. The *LonMarkObject* can report if it is under the range of its acceptable values. **FALSE**. The *LonMarkObject* can not report if it is under the range of its acceptable values. *statusObject* — The *ObjectStatus* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

PingIntervals

The *PingIntervals* object defines the duration of the ping interval for each of the four ping classes. Each device and router defines its ping class using the *PingClass* property. Setting these values appropriately allows you to reduce network traffic. The following table summarizes the *PingIntervals* object.

Description	The ping interval defined for the four ping classes.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None
Methods	None.
Properties	• *ClassId* • *MobileClsPingInterval* • *Parent* • *PermanentClsPingInterval* • *StationaryClsPingInterval* • *TemporaryClsPingInterval*

Methods

The *PingIntervals* object does not contain any methods.

Properties

The *PingIntervals* object contains the following properties:

- *ClassId*
- *MobileClsPingInterval*
- *Parent*
- *PermanentClsPingInterval*
- *StationaryClsPingInterval*
- *TemporaryClsPingInterval*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The

	following value is defined for the *PingInterval* object in the *ConstClassIds* constant: **69 lcaClassIdPingIntervals** *object* The object to be acted on.	
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

MobileClassPingInterval

Summary	The cycle time (in seconds) for pinging devices that may move. This value will be used for all devices and routers that have their *PingClass* property set to **lcaPingClassMobile**. All uninstalled devices are in this category because they are more likely to be removed by the user without informing the network management tool.	
Availability	Local, full, and lightweight clients.	
Syntax	*permPing = piObject.***MobileClsPingInterval**	
	Element	**Description**
	mobilePing	The ping interval (in seconds) for the mobile ping class. This value must be at least 30 seconds, and may not exceed any non-zero value for the stationary class. In addition, the following must be true (excluding **0** values, which means pinging is disabled): MobileClsPingInterval <= TemporaryClsPingInterval <= StationaryClsPingInterval <= PermanentClsPingInterval You must change each of these properties individually; therefore, when you start a transaction, you can set each of these properties to **0**, set the properties to the desired values, and then commit the transaction. Set this property to **0** to disable pinging. The minimum value is **30** seconds. **65535 (-1)** means "do not change."

		The default value is **60** seconds for private media, and **0** seconds for shared media.
	piObject	The *PingIntervals* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.	
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.	
Syntax	*parentObject* = *object*.**Parent**	
	Element	**Description**
	parentObject	The object that is the parent of the specified *object*.
	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

PermanentClassPingInterval

Summary	The cycle time (in seconds) for pinging devices that are expected to never move. This value will be used for all devices and routers that have their *PingClass* property set to **lcaPingClassPermanent**.	
Availability	Local, full, and lightweight clients.	
Syntax	*permPing* = *piObject*.**PermanentClsPingInterval**	
	Element	**Description**
	permPing	The ping interval (in seconds) for the permanent ping class. Set this property to **0** to disable pinging. The default value is **0** seconds.

		The minimum value is **900** seconds. **65535 (-1)** means "do not change".
	piObject	The *PingIntervals* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

StationaryClassPingInterval

The cycle time in seconds for pinging devices which are expected to move rarely. This value will be used for all devices and routers that have their *PingClass* property set to **lcaPingClassStationary**

Summary	The cycle time (in seconds) for pinging devices that are expected to move rarely. This value will be used for all devices and routers that have their *PingClass* property set to **lcaPingClassStationary**.	
Availability	Local, full, and lightweight clients.	
Syntax	*stationPing = piObject.***PermanentClsPingStationary**	
	Element	**Description**
	stationPing	The ping interval (in seconds) for the stationary ping class. **0** seconds means disabled. The minimum value is **300** seconds. **65535 (-1)** means "do not change". The default value is **900** seconds.
	piObject	The *PingInterval* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

TemporaryClassPingInterval

Summary	The cycle time (in seconds) for pinging devices that are expected to move frequently. This value will be used for all devices and routers that have their *PingClass* property set to **lcaPingClassTemporary**. All devices default to this class after installation.	
Availability	Local, full, and lightweight clients.	
Syntax	*tempPing = piObject.***TemporaryClsPingInterval**	
	Element	**Description**

	tempPing	The ping interval (in seconds) for the temporary ping class. Set this property to **0** seconds to disable pinging. The minimum value is **60** seconds. **65535 (-1)** means "do not change". The default value is **120** seconds.
	piObject	The *PingIntervals* object to be acted on.
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.0.	

RecoveryStatus

A *RecoveryStatus* object reports status information about the current phase of OpenLNS database recovery. You can initiate a network recovery with the *PrepareToRecoverFromNetwork* and *RecoverFromNetwork* methods. The process that initiated the network recovery will be blocked until recovery is complete. The *RecoveryStatus* object, unlike most objects, can be accessed during an *OnSystem NssIdle* callback, which allows the client that started the recovery to access the recovery status while recovery is still in progress.

The following table summarizes the *RecoveryStatus* object.

Description	Reports the status of the current phase in an OpenLNS database recovery.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None
Methods	None.
Properties	• *ClassId* • *CurrentPhaseNumber* • *ItemsInPhase* • *NumberPhases* • *Parent* • *PhaseType* • *ProgressIndicator* • *ProgressIndicatorType* • *Status* • *TotalAppDevices* • *TotalChannels* • *TotalConnections* • *TotalNvMts* • *TotalRouters*

Methods

The *RecoveryStatus* object does not contain any methods.

Properties

The *RecoveryStatus* object contains the following properties:

- *ClassId*
- *CurrentPhaseNumber*
- *ItemsInPhase*
- *NumberPhases*
- *Parent*
- *PhaseType*
- *ProgressIndicator*
- *ProgressIndicatorType*
- *Status*
- *TotalAppDevices*
- *TotalChannels*
- *TotalConnections*
- *TotalNvMts*
- *TotalRouters*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *RecoveryStatus* object in the *ConstClassIds* constant: **69 lcaClassIdPingIntervals** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CurrentPhaseNumber

Summary	An Integer indicating the current phase of the database recovery process.
Availability	Local, full, and lightweight clients.
Syntax	*phaseNumValue* = *rsObject*.**CurrentPhaseNumber** **Element** — **Description** *phaseNumValue* — The current phase number as an integer. Valid values from this property are 0 to the value in the *NumberPhases* property. This value can be used in conjunction with the *NumberPhases* property to provide a rough estimate of progress in the database recovery operation. *rsObject* — The *RecoveryStatus* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ItemsInPhase

Summary	A Long indicating the total items to be processed in this phase.
Availability	Local, full, and lightweight clients.
Syntax	*itemsValue* = *rsObject*.**ItemsInPhase** **Element** — **Description** *itemsValue* — The number of items to be processed. The class of object that an 'item' represents is determined by the *ProgressIndicatorType* property. *rsObject* — The *RecoveryStatus* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NumbersPhases

Summary	The number of phases in the database recovery operation. This value can be used in conjunction with the *CurrentPhaseNumber* property to provide a rough estimate in the progress of a database recovery operation.
Availability	Local, full, and lightweight clients.
Syntax	*itemsValue* = *rsObject*.**ItemsInPhase** **Element** — **Description** *numPhaseValue* — The number of phases in the database recovery operation. *rsObject* — The *RecoveryStatus* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PhaseType

Summary	Indicates the current phase of the database recovery process.
Availability	Local, full, and lightweight clients.
Syntax	*phaseTypeValue* = *rsObject*.**PhaseType**

	Element	**Description**
	phaseTypeValue	The current phase. The phase values, which are contained in the *ConstRecoveryPhaseTypes* constant, are as follows: **0 lcaRecoveryPhasePreparation** **1 lcaRecoveryPhaseTopology** **2 lcaRecoveryPhasePhysNode** **3 lcaRecoveryPhaseLogTopology** **4 lcaRecoveryPhaseDeviceAddress** **5 lcaRecoveryPhaseNvSubgroup** **6 lcaRecoveryPhaseNvSubgoupMember** **7 lcaRecoveryPhaseMtSubgroup** **8 lcaRecoveryPhaseSubgroupCleanup** **9 lcaRecoveryPhaseGroupCleanup** **10 lcaRecoveryPhaseCleanupAddrs** **11 lcaRecoveryPhaseGrpOverloading** **12 lcaRecoveryPhaseSystemDevice** **13 lcaRecoveryPhaseConnections** **14 lcaRecoveryPhaseAliasConns** **15 lcaRecoveryPhaseNvSelectorPools** **16 lcaRecoveryPhaseMonitorPoints** **17 lcaRecoveryPhaseCpValues**
	rsObject	The *RecoveryStatus* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ProgressIndicator

Summary	Indicates how much of the database has been processed in this phase. The value in this property is the number of items of the type specified in the *ProgressIndicatorType* property that have been completed during this phase of database recovery. Comparing this value to that of the *ItemsInPhase* property, generates an estimate of phase completion.	
Availability	Local, full, and lightweight clients.	
Syntax	*progIndicValue = rsObject.***ProgressIndicator**	
	Element	**Description**
	progIndicValue	The progress indicator value. The values for this element are contained in the *ConstClassIds* constant. This property will be one of the following values:

	7 lcaClassIdAppDevice **9 lcaClassIdRouter** **18 lcaClassIdConnections** **22 lcaClassIdMessageTag** **24 lcaClassIdNetworkVariable** *rsObject* The *RecoveryStatus* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ProgressIndicatorType

Summary	Indicates the object type by which the status of a database recovery process will be measured. This property determines the units for the *ItemsInPhase* and *ProgressIndicator* properties. The status of a phase in a recovery may be determined by the number of *AppDevice*s, *Routers*, *Connections*, *Network Variables*, or *Message Tags* processed.
Availability	Local, full, and lightweight clients.
Syntax	*progIndicTypeValue = rsObject.ProgressIndicatorType* **Element** — **Description** *progIndicTypeValue* — The measured object type. *rsObject* — The *RecoveryStatus* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Status

Summary	Indicates the status of the database recovery process.
Availability	Local, full, and lightweight clients.

Syntax	*progIndicTypeValue* = *rsObject*.*ProgressIndicatorType* **Element** **Description** *rsObject* The *RecoveryStatus* object to be acted on. *statusValue* The status of the database recovery process. The values for this element, which are contained in the *ConstRecoveryStatus* constant, are as follows: **0 lcaRecoveryStatusNone** **1 lcaRecoveryStatusActive** **2 lcaRecoveryStatusPending** **3 lcaRecoveryStatusComplete**
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TotalAppDevices

Summary	The total number of *AppDevices* discovered during the recovery.
Availability	Local, full, and lightweight clients.
Syntax	*totalDevicesValue* = *rsObject*.**TotalAppDevices** **Element** **Description** *rsObject* The *RecoveryStatus* object to be acted on. *totalDevicesValue* The total number of application devices discovered as a long. A value of -1 indicates that the number is unknown.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TotalChannels

Summary	The total number of *Channels* discovered during the recovery.
Availability	Local, full, and lightweight clients.

Syntax	*totalChannelsValue* = *rsObject*.**TotalChannels** **Element** — **Description** *rsObject* — The *RecoveryStatus* object to be acted on. *totalChannelsValue* — The total number of channels discovered as a long. A value of -1 indicates that the number is unknown.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TotalConnections

Summary	The total number of *Connections* reconstructed during the recovery.
Availability	Local, full, and lightweight clients.
Syntax	*totalConnectionsValue* = *rsObject*.**TotalConnections** **Element** — **Description** *rsObject* — The *RecoveryStatus* object to be acted on. *totalConnectionsValue* — The total number of connections reconstructed as a long. A value of -1 indicates that the number is unknown.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TotalNvMts

Summary	The total number of *Network Variables* and *Message Tags* discovered during the recovery.
Availability	Local, full, and lightweight clients.
Syntax	*totalNvMtsValue* = *rsObject*.**TotalNvMts** **Element** — **Description** *rsObject* — The *RecoveryStatus* object to be acted on. *totalNvMtsValue* — The total number of application devices discovered as a long. A value of -1 indicates that the number is unknown.

Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TotalRouters

Summary	The total number of *Routers* discovered during the recovery.
Availability	Local, full, and lightweight clients.
Syntax	*totalRoutersValue* = *rsObject*.**TotalRouters** **Element** — **Description** *rsObject* — The *RecoveryStatus* object to be acted on. *totalRoutersValue* — The total number of routers discovered as a long. A value of -1 indicates that the number is unknown.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Router

A *Router* object represents a LONWORKS router. The following table summarizes the *Router* object.

Description	Represents a LONWORKS router.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Routers* collection object.
Default Property	*Name* property.
Methods	• *Commission* • *CommissionEx* • *Decommission* • *MoveEx* • *PostMove* • *PreMove* • *Reboot* • *Replace* • *ReplaceEx* • *Reset* • *Test*
Properties	• *AttachmentStatus* • *AuthenticationEnabled* • *BitmapFilePath* • *Class* • *ClassId*

	• *CommissionStatus* • *Description* • *Extensions* • *FarSide* • *Handle* • *IconFilePath* • *InitialAuthenticationKey* • *Location* • *Name* • *NearSide* • *Parent* • *PingClass* • *State* • *Subsystems*

Methods

The *Router* object contains the following methods.

- *Commission*
- *CommissionEx*
- *Decommission*
- *MoveEx*
- *PostMove*
- *PreMove*
- *Reboot*
- *Replace*
- *ReplaceEx*
- *Reset*
- *Test*

Commission

Summary	Associates a *Router* object with a physical router, and loads the router's network image. When commissioning a router while in communication with the near side of the router, the Neuron ID of the *near side* must be set. If not connected to the network, or not in communication with the network, the Neuron ID must be set for the near side and the *far side* of the router. When commissioning is complete, a router is placed in the **lcaStateCnfgOnline** state. Do not read or write to the *State* property of a *Router* in the same explicit transaction with this method. In addition, you must set the *InitialAuthenticationKey* property to the proper value before commissioning an application device or router that has network management authentication enabled, and has been previously commissioned outside of OpenLNS, or commissioned on a different OpenLNS network. In these circumstances, OpenLNS will not be able to communicate with the device or router without knowing its authentication key, since the

	device has authentication enabled. If the *InitialAuthenticationKey* is set prior to commissioning the device or router, OpenLNS will use this key to authenticate messages sent to the device or router during the commissioning process. See the *OpenLNS Programmer's Guide* for more information on the steps you should take when installing devices on a network and commissioning those devices.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Commission** **Element** **Description** *object* The *Router* object to be commissioned.
Added to API	Prior to LNS Release 3.0.

CommissionEx

Summary	Associates a Router object with a physical router, and loads the router's network image. This method is the same as the *Commission* method except that it provides additional options that you can choose from when commissioning the router (see the description of the *options* element for more information).
Availability	Local, full, and lightweight clients.
Syntax	*object*.**CommissionEx** *options* **Element** **Description** *object* The *Router* object to be commissioned. *options* A Long value indicating the options to be used when commissioning the router. This determines whether the changes caused by the commission process will be applied to the physical router if the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**. The values for this element, which are stored in the *ConstCommissionFlags* constant, are as follows: **0 lcaCommissionFlagNone** Updates the device as soon as the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0).** Use this option if you do not want the device to be updated if the network management mode property is set to **lcaMgmtModeDeferConfigUpdates**

	(1). **1 lcaCommissionFlagPropagate** Propagates the device's network image in the OpenLNS database to the physical device when the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1).** If this will cause network inconsistencies, OpenLNS will defer the updates, and the **NS, #4039 lcaErrNsUpdatesDeferred** exception will be thrown. If the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)**, the device's network image in the OpenLNS database will be propagated to the physical device, regardless of whether this option is set.
Added to API	Prior to LNS Release 3.0.

Decommission

Summary	Sets the *NeuronId* property of the Router to "000000000000" (none) and deconfigures the device.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Decommission** **Element** — **Description** *object* — The *Router* object to be decommissioned.
Added to API	Prior to LNS Release 3.0.

MoveEx

Summary	Performs the steps required to move a router to a new channel. This method combines the functions performed by the *PreMove* method and *PostMove* methods, and adds a flag parameter to specify advanced options required for a small set of move operations. Applications should use the *PreMove* and *PostMove* methods whenever possible. You must invoke the *MoveEx* method twice during a move. You first need to invoke it with the **lcaMovePrePhysical** flag (analogous to the *PreMove* call), and then once the router ha sbeen moved, you need to invoke it again with the **lcaMovePostPhysical** flag. You should invoke the *BeginSession* method to begin a

	session before using the *MoveEx* method. This will allow your application to avoid some failure scenarios that may occur when routers are moved.
Availability	Local, full, and lightweight clients.
Syntax	*routerObject.MoveEx newChannelObject, newSubnetObject, flags*

Element	Description
routerObject	The router to be moved from channel to channel.
oldNearChannel	The old near side channel for the router.
oldFarChannel	The old far side channel for the router.
newNearChannel	The new near side channel for the router.
newFarChannel	The new far side channel for the router.
routerType	An Integer value specifying the type of router.
flags	A Long value specifying the movement flags. Multiple options may be specified by logically OR'ing individual flag values. The flag values, which are provided in the *ConstMoveExFlags* constants, are as follows:
	1 lcaMovePrePhysical
	Specified when invoking the *MoveEx* method prior to the physical move
	2 lcaMovePostPhysical
	Specified when invoking the *MoveEx* method following the physical move.
	8 lcaMoveOnline
	Indicates that moved routers are to be left online (default for routers).
	16 lcaMoveOffline
	Indicates that moved routers are to be left offline (default for routers).
	32 lcaMoveRestore
	Indicates that moved routers will be restored to their original online/offline state. If a router is power cycled or reset as part of the

	move, this information is lost, and the router will be put online.
Added to API	Prior to LNS Release 3.0.

PostMove

Summary	Completes the move of a router from one channel to another. As part of the move, the router's network address may change; however, the router's *Handle* property cannot change. To move a router from one channel to another, follow these steps: 1. Invoke the *BeginSession* method to begin a session. This will allow your application to avoid some failure scenarios that may occur when routers are moved. 2. Invoke the *PreMove* method. The target channel is validated and the router is deconfigured. However, the router still appears in the database as residing on the original channel. 3. Physically move the router from one channel to another. 4. If the router being moved uses authentication, the *PostMove* method should be called in a different session than the *PreMove* method. This means you should call the *EndSession* method to end the session begun in step 1, and then call *BeginSession* to begin a new session. 5. Invoke the *PostMove* method. This method changes the router's channel assignment, does any necessary rebinding, and updates the router's configuration. If you use explicit transactions during this procedure, make sure that there are separate ones for the *PreMove* and *PostMove* steps.
Availability	Local, full, and lightweight clients.
Syntax	*object.PostMove* **Element** — **Description** *object* — The *Router* object to be moved from one channel to another.
Added to API	Prior to LNS Release 3.0.

PreMove

Summary	Prepares a router for movement from one channel (or subnet) to another. When you call this method, you must specify the new channels (*newNearChannel* and *newFarChannel* elements). If this method is called while not attached to the network, this message will throw an exception. Ignore it and call the

	PostMove method to move a router while in engineered mode. See the *PostMove* method for more information on the steps required to move a router.
Availability	Local, full, and lightweight clients.
Syntax	*routerObject.PreMove newNearChannel, newFarChannel* **Element** — **Description** *routerObject* — The router to be moved from channel to channel. *newNearChannel* — The new near side channel for the router. *newFarChannel* — The new far side channel for the router.
Added to API	Prior to LNS Release 3.0.

Reboot

Summary	You can use this method to reboot a router. Rebooting a router may destroy its communication parameters or otherwise make the router unrecoverable. The result of rebooting a router depends on the firmware state and reboot options specified at the time of manufacture.
Availability	Local, full, and lightweight clients.
Syntax	*object.***Reboot** **Element** — **Description** *object* — The *Router* object to be rebooted.
Added to API	Prior to LNS Release 3.0.

Replace

Summary	Replaces one router with another. This method is typically used to effect repair operations. The new router receives the same network address and routing configuration as the old router. To automatically load the old router's configuration into the new router, use the *ReplaceEx* method. The *ReplaceEx* method performs the same function as the *Replace* method; however, it provides options you can use to determine how the configuration on the old router will be managed during the replacement. To specify the new router, you only need to set the *NeuronId* property of the *RouterSide* object that corresponds to the near side of the new router. When the replace method is complete, a new router router is placed in the online state (*lcaCnfgOnline*). **Note**: Do not read or write the *State* property of the router in

	the same explicit transaction as the invocation of this method.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Replace** **Element** **Description** *object* The *Router* to be replaced.
Added to API	Prior to LNS Release 3.0.

ReplaceEx

Summary	Replaces one router with another. This method is typically used to effect repair operations. The new router receives the same network address and routing configuration as the old router. This method is similar to the *Replace* method except that it provides additional options you can use. See the description of the *options* element for more information. See the *Replace* method for more general information on replacing routers.
Availability	Local, full, and lightweight clients.
Syntax	o*bject*.**ReplaceEx** *flags* **Element** **Description** *object* The Router object to be replaced. *flags* The options which apply to this *Replace* operation. The possible values for this element, which are contained in the *ConstReplaceFlags* constant, are as follows: **0 No options** Enter this value if do not want to use any options. **8 lcaReplaceFlagPropagateUpdates** Propagates the network image contained in the database to the new router, and deconfigures the old router, even if the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)** when you call the *ReplaceEx* method. If any of these changes will cause network inconsistencies, OpenLNS will defer the updates, and the NS, #4039 *lcaErrNsUpdatesDeferred* exception will be thrown. Those updates will not be propagated until the network management mode is set back to

	lcaMgmtModePropagateConfigUpdates (0). If the network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)** when you call the *ReplaceEx* method, the network image contained in the database will be propagated to the new router and the old router will be deconfigured, regardless if this option has been set.
Added to API	Prior to LNS Release 3.0.

Reset

Summary	Sends a reset command to the router.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**Reset** **Element** **Description** *object* The *Router* object to be reset.
Added to API	Prior to LNS Release 3.0.

Test

Summary	Tests a router. You must set the router's Neuron ID before invoking this method. You can use this method to verify that a router is able to communicate on the network, and that a subset of its configuration matches the information contained in the OpenLNS database. To pass the test, a router with the expected Neuron ID must exist on the network, and respond to queries. If the router is installed, it must contain the expected domain/subnet/node address, and respond to subnet/node messages on that address. It must also contain the expected program ID, and the expected network management authentication setting (enabled or disabled). If network management authentication is enabled on the router, it must contain the correct system authentication key. Furthermore, the test verifies that there are no other nodes that respond with the tested nodes subnet/node address. The results of the test are contained in the Router object's *LastTestInfo* property. This service analyzes the result with the assumption that the node's network image is up-to-date; therefore, ensure that the node is up-to-date by invoking this service only while the *MgmtMode* property is set to **lcaMgmtModePropagateConfigUpdates (0)**. This is because discrepancies between the OpenLNS database and the current configuration of the router on the network are

	normal, and can be expected while the management mode is **lcaMgmtModeDeferConfigUpdates (1)**. These discrepancies will typically be resolved automatically when the system manage mode has been changed to **lcaMgmtModePropagateConfigUpdates (0)**. For more information on the *Test* method, see the *Detecting Devices and Detecting Device Failures* section in the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*testStatus* = *routerToTest*.**Test** **Element** — **Description** *testStatus* — An Integer value with the results returned by the test. The enumerated values for *testStatus*, which are contained in the *ConstTestResults* constant, are as follows: **0 lcaTestResultGood** The router passed all applicable tests. **1 lcaTestResultComm** The OpenLNS Server was unable to communicate with the router using either Neuron ID or its subnet/node addressing. The *NetworkServiceDevice* might not be attached to the network, the *NetworkServiceDevice* might be unconfigured, the target router might not be attached to the network, the target router may be powered off, or the target router may be faulty. **2 lcaTestResultCommNeuronId** The OpenLNS Server was unable to communicate with the router using Neuron ID addressing. Because the router has not been added or defined, the OpenLNS Server could not test the router using subnet/node ID addressing. The OpenLNS Server might not be attached to the network, the *NetworkServiceDevice* might be unconfigured, the target router might not be attached to the network, the target router may be powered off, or the target router may be faulty. **3 lcaTestResultCommNeuronId Verified** The OpenLNS Server was able to communicate with the router using

		subnet/node addressing, but could not communicate with the router using Neuron ID addressing. However, through the use of subnet/node addressed messages, the OpenLNS Server has verified that the router contains the expected Neuron ID. This might be caused by intermittent router or channel failures. You may want to retry this method to see if the error persists. **4 lcaTestResultCommSnode** The OpenLNS Server was able to communicate with the router using Neuron ID addressing, but could not communicate with the router using subnet/node addressing. The OpenLNS Server was unable to verify whether the router has been configured with the proper domain/subnet/node address. The router may have reconfigured itself, the router may have been reconfigured by another network management tool, or the router may be faulty. The router may be restored using the *Commission* method. **5 lcaTestResultSnodeVerified** The OpenLNS Server was able to communicate with the router using Neuron ID addressing, but could not communicate with the router using subnet/node addressing. However, through the use of Neuron ID addressed messages, The OpenLNS Server has verified that the router contains the expected subnet/node address. This error can occur if the target router is in an unconfigured state. The router may need to be recomissioned or it may be in the middle of a two-phase move. It also might be caused by intermittent router or channel failures. You may want to retry this method to see if the error persists. **6 lcaTestResultDuplicateSnode** The OpenLNS Server was able to communicate with the router using Neuron ID addressing. However, when using subnet/node addressing, the responding router contains a different Neuron ID. This failure indicates that multiple routers are configured with the same domain/subnet/node address. This

		could be caused by attaching a new router to the network which was previously configured as part of another network or by two networks sharing the same media and domain ID. This condition could also occur if the unexpected router was supposed to be removed from the system, but the OpenLNS Server was unable to update the router's network image during a *Remove* method; meanwhile, the removed subnet/node address has been reused for the router being tested. **7 lcaTestResultMismatchDomain** The OpenLNS Server was able to communicate with the router using Neuron ID addressing but could not communicate with the router using subnet/node addressing. Using Neuron ID addressing, the OpenLNS Server found that the domain ID configured in the router does not match the database. This result will be returned if the router's domain address has not be configured or the router has become unconfigured due to a checksum error. **8 lcaTestResultMismatchNeuronId** The OpenLNS Server was unable to communicate with the router using Neuron ID addressing. However, the router that responded to the subnet/node addressed test message contains a different Neuron ID. It appears that the subnet/node address is configured in the responding router but not the tested router. One possible cause is that the tested router is supposed to replace the responding router, but the network images in both routers have not been updated yet. You can resolve this problem by physically removing the obsolete router. **9 lcaTestResultMismatchSnode** The OpenLNS Server was able to communicate with the router using Neuron ID addressing, but could not communicate with the router using its subnet/node addressing. Through the use of Neuron ID addressed messages, the NSS has found that the subnet/node address configured in the router does not

		match the database. This result will be returned if the router's domain address has not been configured. **10 lcaTestResultNoNeuronId** The router has not been assigned a Neuron ID. **11 lcaTestResultMismatchProgramId** The OpenLNS Server was able to communicate with the router using both Neuron ID and subnet/node addressing. However, the router does not contain the expected program ID. The router's program ID may have been changed by its application program. Host routers can modify the program ID of their attached network interface. The router should be Removed and Added. **12 lcaTestResultCommSnodeNotVerified** The OpenLNS Server was able to communicate with the router using Neuron ID addressing, but could not communicate with the router using subnet/node addressing. The OpenLNS Server did not attempt to verify that the router has been configured with the proper domain/subnet/node address because the router is currently authenticated, and reading the address would result in transmitting the key over the network. The router may have reconfigured itself, the router may have been reconfigured by another network management tool, or the router may be faulty. The router may be restored using the *Commission* method. **13 lcaTestResultAuthEnabled** The OpenLNS Server was able to communicate with the router using both Neuron ID addressing and subnet/node addressing. However, the router has network management authentication *enabled* despite the fact that the router's *AuthenticationEnabled* property is set to FALSE. The router may have enabled network management authentication itself, the router may have been

		reconfigured by another network management tool, or the router may be faulty. It may be possible to restore the router using the *Commission* method. **14 lcaTestResultAuthDisabled** The OpenLNS Server was able to communicate with the router using both Neuron ID addressing and subnet/node addressing. However, the router has network management authentication *disabled* despite the fact that the router's AuthenticationEnabled property is set to TRUE. The router may have disabled network management authentication itself, the router may have been reconfigured by another network management tool, or the router may be faulty. The router may be restored using the *Commission* method. **15 lcaTestResultKeyMismatch** The OpenLNS Server was able to communicate with the router using both Neuron ID addressing and subnet/node addressing. The router has network management authentication enabled and the node's AuthentictionEnabled property is set to TRUE. However, the router does not contain the current system authentication key. The router may have changed its authentication key itself, the router may have been reconfigured by another network management tool, or the router may be faulty. It may be possible to restore the router using the *Commission* method. **16 lcaTestResultInterfaceFailure** The OpenLNS Server was unable to communicate with the OpenLNS network interface. The OpenLNS network interface may have become disconnected or faulty. Exit all OpenLNS applications and perform diagnostics on the OpenLNS network interface using the LONWORKS Interfaces Control Panel application. **17 lcaTestResultInterfaceNotOnline** The OpenLNS network interface that the OpenLNS Server is attempting to use is not Online. Recommission the NetworkServiceRouter of the *System*

		object by calling the `System.NetworkServiceRouter.Router.Commission` method, and make sure that the *State* property of the *NetworkServiceDevice* object (*NetworkServiceDevice.Router.State*) is set to lcaOnline. **18 lcaTestResultInterfaceConfigError** The OpenLNS network interface that the OpenLNS Server is attempting to use is not property configured. Recommission the *NetworkServiceDevice* of the *System* object by calling the `System.NetworkServiceDevice.Router.Commission` method.
	routerToTest	The *Router* to be tested.
Added to API	Prior to LNS Release 3.0.	

Properties

The *Router* object contains the following properties:

- *AttachmentStatus*
- *AuthenticationEnabled*
- *BitmapFilePath*
- *Class*
- *ClassId*
- *CommissionStatus*
- *Description*
- *Extensions*
- *FarSide*
- *Handle*
- *IconFilePath*
- *InitialAuthenticationKey*
- *Location*
- *Name*
- *NearSide*
- *Parent*
- *PingClass*
- *State*
- *Subsystems*

AttachmentStatus

Summary	Indicates whether the device is attached and in the proper state.
Availability	Local, full, and lightweight clients.

Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *attachmentStatus* — The attachment status of the router. The values for this element, which are stored in the *ConstDeviceAttachmentStatus* constant, are as follows: **0 lcaDeviceAttached** Indicates that the router that triggered the event is now attached and in the proper state. **1 lcaDeviceInImproperState** Indicates that the router that triggered the event is now attached but not in the proper state. **2 lcaDeviceNotAttached** Indicates that the router that triggered the event is now not attached. *Object* — The *Router* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

AuthenticationEnabled

Summary	Indicates whether a router uses network management authentication. The following sections describe how to enable and disable authentication. **Enabling Authentication** To enable authentication for any of the devices in a system, follow these steps: 1. Set the system's authentication key using the *System* object's *AuthenticationKey* property. 2. Set the *AuthenticationEnabled* property of the *AppDevice* object that represents the *System* object's *NetworkServiceDevice* object to True. This enables authentication for all Network Service Devices on the system. As a result, the authentication key used by each Full and Lightweight client application must match the authentication key established in Step 1 the next time any of those applications opens the network. Note that if you have set the authentication key used by a Full client's network interface to match the authentication key established in step 1, then other clients can use that connection without re-specifying the key, as long the first

<table>
<tr><td></td><td>Full client remains connected to the server.
3. Set the AuthenticationEnabled property to True the AppDevice objects that will use authentication.
Disabling Authentication
To disable authentication for all the devices in a system, follow these steps:
1. Set the AuthenticationEnabled property to False on the AppDevice object that represents the System object's NetworkServiceDevice. This disables authentication for all application devices, routers and Network Service Devices operating on the system.
2. Set the System object's AuthenticationKey property to ffff ffff ffff to indicate that authentication is disabled.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>authEnabled = Object.AuthenticationEnabled
Element — Description
authEnabled — Boolean value indicating whether a router uses network management authentication.
If the SecurityLevel property of the System object is set to lcaSecurityLevelKeyDistributionEnabled, setting this property to True enables network management authentication and installs the system key in the router.
If the SecurityLevel is not lcaSecurityLevelKeyDistributionEnabled, setting this property to True implies that network management authentication will be enabled by the ObjectServer, but the system key will be installed in the router side by some external means.
Setting this property to False results in removing the system key from the router side and disabling network management authentication on the router side.
Only application devices whose AuthenticationEnabled property is set to True are permitted to participate in authenticated connections. If an application device participates in authenticated connections, you cannot set this property to False.
Object — The object to be acted on.</td></tr>
<tr><td>Data Type</td><td>Boolean.</td></tr>
<tr><td>Read/Write</td><td>Read/write.</td></tr>
</table>

Added to API	Prior to LNS Release 3.0.

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.
Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** — **Description** *bmpFilePath* — The bitmap path and file name. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).
Added to API	Prior to LNS Release 3.0.

Class

Summary	Identifies the router class (repeater, bridge, learning router or configured router).
Availability	Local, full, and lightweight clients.
Syntax	*classValue* = r*outerObject*.**Class** **Element** — **Description** *classValue* — The class of the router. The valid class values, which are contained in the *ConstRouterClasses* constant, are as follows: **0 lcaConfiguredRouter** **1 lcaLearningRouter (*see notes)** **2 lcaRepeater** **3 lcaBridge** **4 lcaPermanentRepeater** **5 lcaPermanentBridge** **Notes**: The **lcaLearningRouter (1)** value is not supported by OpenLNS. If you write the

		lcaLearningRouter (1) value to this property, it will automatically be converted to **lcaConfiguredRouter (0)**. A permanent repeater or bridge cannot be changed to a non-permanent class. Similarly, a non-permanent class cannot be changed to a permanent repeater or bridge. *routerObject* The *Router* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Router* object in the *ConstClassIds* constant: **9 lcaClassIdRouter** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

CommissionStatus

Summary	Stores the commission status of a router. After adding a new router and setting the *NeuronId* property, but before the *Commission* method is called, this property will be set to **lcaCommissionUpdatesPending (1)**. After a successful commission, this property will be **lcaCommissionUpdatesCurrent (0)**.
Availability	Local, full, and lightweight clients.

Syntax	*commStatusValue* = *object*.**CommissionStatus** **Element** — **Description** *object* — The router object to be acted on. *commStatusValue* — The commission status of this router. The values for this element, which are stored in the *commStatusValue* constant, are as follows: **0 lcaCommissionUpdatesCurrent** No outstanding commission updates are pending. **1 lcaCommissionUpdatesPending** Commission updates are currently pending, or in progress. When database changes are made that affect a device's configuration, this value represents the commission status of the device. The commission status will be changed to **lcaCommissionUpdatesCurrent (0)** when the changes are successfully propogated to the device, or to **lcaCommissionUpdatesFailed (2)** if there is a failure to propogate the changes. **2 lcaCommissionUpdatesFailed** Commission updates are currently pending or in progress, and the most recent update attempt failed.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *Router* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *Router* object. *object* — The *Router* object to be acted on.

Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *Router*. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.*Extensions* **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FarSide

Summary	Returns the far side *RouterSide* object for the *Router* device, relative to the system's *NetworkServiceDevice* that is used by the OpenLNS Server computer. The *NetworkServiceDevice* can be accessed through the N*etworkServiceDevice* property of the *System* object. **Note**: The *FarSide* and *NearSide* properties are adjusted by OpenLNS automatically whenever the system's *NetworkServiceDevice* is moved. For more information on moving Network Service Devices, see the Network Management: Advanced Topics chapter in the OpenLNS Programmer's Guide.
Availability	Local, full, and lightweight clients.
Syntax	*routerSideObj* = *routerObj*.**NearSide** **Element** — **Description** *routerSideObj* — The router's near side. *routerObject* — The *Router* object to be acted upon.
Data Type	*RouterSide* object.

Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *Router* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property and may also be read using the *Index* property. You can also use the *ItemByHandle* method to retrive *Router* objects.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the object. *object* — The *Router* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (*.**ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** — **Description** *IconFilePathFileName* — Icon file and path name *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

InitialAuthenticationKey

Summary	Contains the initial authentication key to be used when commissioning a router. This property must be set prior to commissioning a router that has network management authentication enabled, and has been previously commissioned outside of OpenLNS, or commissioned on a different OpenLNS network. In these circumstances, OpenLNS will not be able to communicate with the router without knowing its authentication key because the router has authentication enabled. If the *InitialAuthenticationKey* is set prior to commissioning the router or router, OpenLNS will use this key to authenticate messages sent to the router during the commissioning process. This property does not affect the key stored in the router after a successful commission. Once the router has been commissioned, its authentication key will either be set to FFFFFFFFFFFF or the OpenLNS system authentication key, as determined by the router's *AuthenticationEnabled* property.
Availability	Local, full, and lightweight clients.
Syntax	*authenticationKey* = *object*.*InitialAuthenticationKey* **Element** — **Description** *authenticationKey* — The authentication key to be used when commissioning the router. *object* — The *Router* object being acted upon.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Location

Summary	Contains the value of the specified Router's six byte location as a hex string. This property must contain a 12 digit hex string that is a valid hexadecimal value. The only valid characters are 0-9 and A-F. For example, "0000AC43F1B6" is a valid value. The *Location* property allows you to read the *Router's* location from the OpenLNS database.
Availability	Local, full, and lightweight clients.
Syntax	*locationValue* **=** *object*.**Location** *object*.**Location =** *locationValue* **Element** — **Description** *locationValue* — The location as read from the router as a hex string. *object* — The *Router* object to be acted on.

Data Type	String (6 bytes).
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NearSide

Summary	Returns the near side *RouterSide* object for the *Router* device, relative to the system's *NetworkServiceDevice* that is used by the OpenLNS Server computer. The *NetworkServiceDevice* can be accessed through the N*etworkServiceDevice* property of the *System* object. **Note**: The *FarSide* and *NearSide* properties are adjusted by OpenLNS automatically whenever the system's *NetworkServiceDevice* is moved. For more information on moving Network Service Devices, see the Network Management: Advanced Topics chapter in the OpenLNS Programmer's Guide.
Availability	Local, full, and lightweight clients.
Syntax	*routerSideObj* = *routerObj*.**NearSide** **Element** — **Description** *routerSideObj* — The router's near side. *routerObject* — The *Router* object to be acted upon.
Data Type	*RouterSide* object.

Read/Write	Read-only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy. For example, an *AppDevice* object's parent can be a *NetworkServiceDevice* object or a *AppDevices* collection object
Availability	Local, full, lightweight, and independent clients. Note that not all objects that contain this property are available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

PingClass

Summary	Determines the frequency with which a router is tested (pinged) to determine if it is still attached to the network. This property classifies routers based on the probability that the router may be detached. The higher the probability, the more frequently the router will be pinged. The Object Server assumes a router to be detached if it cannot communicate with that router three consecutive times.
Availability	Local, full, and lightweight clients.
Syntax	*pingClassValue* = *Object.PingClass* **Element** — **Description** *Object* — The router object to be acted on. *pingClassValue* — The ping class of this object. The valid values for this element, which are provided in the *ConstPingClass* constant, are as follows: **0 lcaPingClassDefault** If this value is written to the *PingClass*

		property, OpenLNS will use the default **lcaPingClassStationary (3)** value. **1 lcaPingClassMobile** Class for nodes which move frequently. **2 lcaPingClassTemporary** Class for temporary nodes. **3 lcaPingClassStationary** Class for nodes which rarely move. This is the default value. **4 lcaPingClassPermanent** Class for nodes which never move. **Note**: You change the ping interval that applies to each class with the *System* object's *PingIntervals* property.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

State

Summary	Describes the state of a router. You cannot set the state of a router in the same transaction as an *Add*, *Commission*, or Replace method. You cannot set the state of a router during the definition phase of the predefined components installation scenario. The state change may fail if the router has not been completely updated due to a previous communication problem or if the router has not been commissioned. If a failure occurs, use the *Commission* method to force an update of the router. *Routers* are taken offline while they are being configured or commissioned.	
Availability	Local, full, and lightweight clients.	
Syntax	*stateValue* = *object*.*State*	
	Element	**Description**
	stateValue	The state of the router (online or offline). The enumerated values for this property, which are stored in the *ConstRouterStates* constant, are as follows: **Note:** The only two values that may be written to this property are **lcaStateCnfgOnline**(4) and **lcaStateSoftOffline**(12). All other

		properties are read-only. **2 lcaStateUncnfg** The application is loaded but the configuration is either not loaded, being reloaded, or deemed corrupted due to a configuration checksum error. A Neuron Chip also can make itself unconfigured by calling the Neuron C function **go_unconfigured()**. The router's service LED flashes at a one second rate in this state. **3 lcaStateNoApplUncnfg** No application is loaded yet, the application is in the process of being loaded, or the application has been deemed corrupted due to an application checksum error or signature inconsistency. The application does not run in this state. The router's service LED is steadily on in this state. **4 lcaStateCnfgOnline** Normal router state. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The router's service LED is off in this state. **6 lcaStateCnfgOffline** Application loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The router's service LED is off in this state. **12 lcaStateSoftOffline** The router has an application, is configured, and is soft-offline. It will go online when it is reset or when requested to go online. The router's service LED is off in this state. **140 lcaStateCnfgBypass** The application confirmed the offline request, but is still running (bypass mode). The router's service LED is off in this state.
	object	The router to be acted on.

Data Type	Integer.
Read/Write	Read /write.
Added to API	Prior to LNS Release 3.0.

Subsystems

Summary	Contains the *Subsystems* collection object associated with the specified Router. A *Subsystem* object can in turn contain a collection of *Subsystems*. Subsystems collection objects accessed through *Router* objects represent the Subsystems that contain the specified *Router*.
Availability	Local, full, and lightweight clients.
Syntax	*subsystemCollection* = *object*.**Subsystems** **Element** — **Description** *subsystemCollection* — Subsystems collection associated with the *object*. *object* — The *Router* object.
Data Type	Subsystems collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Routers

The *Routers* object contains a collection of *Router* objects. The following table summarizes the *Routers* object.

Description	A collection of *Router* objects.
Added to API	LNS Release 3.0.
Accessed Through	*Channel* object. *Subsystem* object.
Default Property	*Item*
Methods	• *Add* • *AddEx* • *AddReference* • *ItemByHandle* • *ItemByNeuronID* • *Remove* • *RemoveEx*
Properties	• *ClassId* • *Count* • *Item* • *_NewEnum*

Methods

The *Routers* object contains the following methods.

- *Add*
- *AddEx*
- *AddReference*
- *ItemByHandle*
- *ItemByNeuronID*
- *Remove*
- *RemoveEx*

Add

Summary	Defines a new *Router* object. A router consists of two halves, *NearSide* and *FarSide*. This method adds a router to the system, but does not install the physical router. The *Commission* method is used to install the physical router after that router is defined using this method. Adding a router to the system is a two-step operation: definition and commissioning. 1. The *Add* method is used to define the router. 2. The *Commission* method is used to write the network image on the physical router, including its network addresses (one for each side). The *Add* method can be used whether OpenLNS is physically attached to the network or not, and while the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0)** or **lcaMgmtModeDeferConfigUpdates (1)**. To modify router buffers, the router should be added using the *AddEx* method. See the *BufferConfiguration* object for more information.
Availability	Local, full, and lightweight clients.
Syntax	*routerObject* = *routersColl*.**Add**(*routerName, nearChannel, farChannel, routerType*) **Element** — **Description** *routerObject* — The newly defined *Router* object. *routersColl* — The *Routers* collection object. *routerName* — A String containing the name of the new Router. *nearChannel* — The Channel object for the router's near channel. *farChannel* — The Channel object for the router's far channel.

	routerType	An Integer value indicating the router type. The valid class values, which are contained in the *ConstRouterClasses* constant, are as follows: **0 lcaConfiguredRouter** **1 lcaLearningRouter (*see notes)** **2 lcaRepeater** **3 lcaBridge** **4 lcaPermanentRepeater** **5 lcaPermanentBridge** **Notes**: The **lcaLearningRouter (1)** value is not supported by OpenLNS. If you write the **lcaLearningRouter (1)** value to this property, it will automatically be converted to **lcaConfiguredRouter (0)**. A permanent repeater or bridge cannot be changed to a non-permanent class. Similarly, a non-permanent class cannot be changed to a permanent repeater or bridge.
Added to API	Prior to LNS Release 3.0.	

AddEx

Summary	This method is identical to the *Add* method of the *Routers* collection, but it allows you to set additional options when adding a router to the collection. You must set the Neuron ID of the *Router* immediately after adding it with this method, but before you perform any other router operations. You can this by writing to the *NeuronId* property of the router's near side *RouterSide* object, which is accessed through the router's *NearSide* property.
Availability	Local, full, and lightweight clients.
Syntax	*routerObject* = *routersColl*.**AddEx**(*routerName, nearChannel, farChannel, routerType, flags)* **Element** — **Description** *routerObject* — The newly defined *Router* object. *routersColl* — The *Routers* collection object. *routerName* — A String containing the name of the new *Router*.

	nearChannel	The Channel object for the router's near channel.
	farChannel	The Channel object for the router's far channel.
	routerType	An Integer value indicating the router type. The valid class values, which are contained in the *ConstRouterClasses* constant, are as follows: **0 lcaConfiguredRouter** **1 lcaLearningRouter (*see notes)** **2 lcaRepeater** **3 lcaBridge** **4 lcaPermanentRepeater** **5 lcaPermanentBridge** **Notes**: The **lcaLearningRouter (1)** value is not supported by OpenLNS. If you write the **lcaLearningRouter (1)** value to this property, it will automatically be converted to **lcaConfiguredRouter (0)**. A permanent repeater or bridge cannot be changed to a non-permanent class. Similarly, a non-permanent class cannot be changed to a permanent repeater or bridge.
	flags	A Long value specifying the add options for this router. The possible values of this element, which are defined in the *ConstRouterFlags* constant, are as follows: **0 lcaRouterFlagNoSplit** The router should be added normally. This method behaves identically to the *Routers* collection's *Add* method if the *flags* parameter is set to **0**. **1 lcaRouterFlagSplit** The router being added splits a previously defined channel into two pieces. When this option is specified, the NSS splits the channel and automatically relocates nodes which are to end up on the far side of the router. A permanent bridge or permanent repeater cannot be used to split a channel. A router cannot perform a split if it would result in a subnet broadcast connection spanning the router. This option only applies if the

	network management mode is set to **lcaMgmtModePropagateConfigUpdates (0)**.
Added to API	Prior to LNS Release 3.0.

AddReference

Summary	Adds a *Router* object reference to a *Routers* collection. This method can be used to add an existing *Router* to multiple *Subsystem* objects (by invoking this method on the *Routers* collection in those *Subsystem* objects). For example, an application might contain both a logical hierarchy for the system (where each subsystem represents a function, such as lighting control or the first stage of a batch process) and a physical hierarchy for the system (where each subsystem represents a physical place such as a room or cell). This method allows *Routers* to be placed within both hierarchies in the appropriate subsystems. When initially defining a router, it is first added to a single subsystem. References to the router may then be added to other subsystems. The router is not deleted from the OpenLNS database or decommissioned until all references have been deleted. The router's association with the first subsystem is also treated as a reference, so it may be removed from its initial subsystem at any time. If *AddReference* is used to "add" a router discovered in one of the Discovered.<xxx> subsystems, invoking this method will cause the router to be removed from its original discovered subsystem.
Availability	Local, full, and lightweight clients.
Syntax	*collection*.**AddReference** *object* **Element** — **Description** *collection* — The *Routers* collection to gain the reference. *object* — The *Router* object to be added.
Added to API	Prior to LNS Release 3.0.

ItemByHandle

Summary	Retrieves a *Router* object, specified by its handle, from a *Routers* collection. The *Router* object to be retrieved must be specified by its handle.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.**ItemByHandle**(*handle*) **Element** — **Description** *object* — The *Router* object retrieved from the

		Routers collection.
	coll	The *Routers* collection object.
	handle	A Long value specifying the handle of the *Router* object to be retrieved.
Added to API	LNS Release 3.0.	

ItemByNeuronID

Summary	Retrieves a *Router* object, specified by its *NeuronID* property, from a *Routers* collection.	
Availability	Local, full, and lightweight clients.	
Syntax	*object* = *coll*.**ItemByNeuronId**(*neuronId*)	
	Element	**Description**
	object	The *Router* retrieved from the collection.
	coll	The *Routers* collection object.
	neuronId	A String specifying the Neuron ID of the *Router* object to be retrieved.
Added to API	LNS Release 3.0.	

Remove

Summary	Removes a router from a subsystem. If the router is not a member of any other subsystems, then it is removed from the system and the network. A router is removed from a subsystem by removing the corresponding *Router* object from the *Routers* collection object owned by that *Subsystem* object. If the router is in any other subsystems, then nothing further is done. If the router in not a member of any other subsystem, then the router is completely removed from the system. The router is removed from all connections, removed from the system domain, and placed in the unconfigured state; the router's channel ID is set to 0. No other changes are made in the router's network image. The NSS *Handle* is also freed and is available for use by the Object Server. A *Router* object can only be removed from a "regular" subsystem. The *Routers* collection object's *Remove* method cannot be used on the *All*, *Discovered.Installed*, or *Discovered.Uninstalled* subsystems.	
Availability	Local, full, and lightweight clients.	
Syntax	*RoutersColl*.**Remove** *indexName*	
	Element	**Description**
	RoutersColl	The *Router* collection object containing

		the router to be removed.
	indexName	A Long value specifying the collection index of the *Router* object to remove, or a String value specifying the name of the *Router* object to remove.
Added to API	Prior to LNS Release 3.0.	

RemoveEx

Summary	This method is identical to the *Remove* method of the *Routers* collection, but it allows you to set additional options when removing a router from the collection.
Availability	Local, full, and lightweight clients.
Syntax	*routersColl*.**RemoveEx** *index, flags* **Element** — **Description** *routersColl* — The *Routers* collection object. *index* — A Long value specifying the collection index of the router to remove. *flags* — A Long value specifying the removal options for this router. The possible values of this element, which are defined in the *ConstRouterFlags* constant, are as follows: **0 lcaRouterFlagNoMerge** The router should be removed normally. **1 lcaRouterFlagMerge** Allows a router to be removed even though the far side channel has nodes attached. If this option is specified, the near side channel will be removed. All devices on the near side channel will be moved to the far side channel. The application is responsible for applying the appropriate label to the resulting channel. You can apply this option to any two routers, even if they do not use the same transceiver type. However, devices using the remaining channel after the merge will be unable to communicate due to the incompatibility between the two routers. As a result, before merging two routers with this option, you should make sure the routers use the same transceiver type. This option only applies if the *network management mode* property is set to **lcaMgmtModePropagateConfigUpdate**

	s (0).
Added to API	Prior to LNS Release 3.0.

Properties

The *Routers* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Routers* object in the *ConstClassIds* constant: **10 lcaClassIdRouters** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer.

	object The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *Router* object from a *Routers* collection. You can retrieve a *Router* object from its *Routers* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *Router* object in *Routers* collections with the *Name* property by passing the object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *Router* object retrieved from the collection. *collObject* — The *Routers* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *Router* object to be retrieved. *stringExpression* — A string type specifying the name of the *Router* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic,

	you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

RouterSide

A *RouterSide* object represents one channel side of a *Router*. The *FarSide* and *NearSide* properties of a Router object each contain a *RouterSide* object. The following table summarizes the *RouterSide* object.

Description	Represents one channel side of a *Router* .
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Router* object.
Default Property	None.
Methods	• *ClearStatus*
Properties	• *BufferConfiguration* • *Channel* • *ClassId* • *DetailInfo* • *LastTestInfo* • *NeuronId* • *NodeId* • *Parent* • *PendingNeuronId* • *Priority* • *ProgramId* • *State* • *SubnetId* • *Subnets*

Methods

The *RouterSide* object contains the following method.

- *ClearStatus*

ClearStatus

Summary	Clears the status information stored in the router side. The clear status method causes a LonTalk Clear *Status* network diagnostic message to be sent to the specified router side. This clears the error log, last reset cause, and communication counters, which are generally accessed by reading the router side's *DetailInfo* object.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**ClearStatus** **Element** **Description** *object* The *RouterSide* object.
Added to API	Prior to LNS Release 3.0.

Properties

The *RouterSide* object contains the following properties.

- *BufferConfiguration*
- *Channel*
- *ClassId*
- *DetailInfo*
- *LastTestInfo*
- *NeuronId*
- *NodeId*
- *Parent*
- *PendingNeuronId*
- *Priority*
- *ProgramId*
- *State*
- *SubnetId*
- *Subnets*

BufferConfiguration

Summary	Contains the *BufferConfiguration* object associated with the specified *RouterSide* object. The property values contained within the *BufferConfiguration* object are not propagated to the network when individually set. To have the OpenLNS Server update the new values on the network, the *BufferConfiguration* property must be set with the modified *BufferConfiguration* object.

	The maximum buffer size for a router side is determined when the router's external interface file is imported. No changes may be made to a *RouterSide* object's *BufferConfiguration* property, which would cause the total memory usage value to be exceeded. If the router was defined without specifying an external interface, OpenLNS will not be able to determine much memory the router has, and will not allow any changes to the *BufferConfiguration* property. To change the maximum buffer size, you can specify a different external interface file that allows for greater memory for a router. For more information on this procedure, see the *BufferConfiguration* object.
Availability	Local, full, and lightweight clients.
Syntax	*bcObject* = *rsObject*.**BufferConfiguration** **Element** — **Description** *bcObject* — The *BufferConfiguration* object retrieved from the object. *rsObject* — The *RouterSide* object to be acted on.
Data Type	*BufferConfiguration* Object.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Channel

Summary	Contains the *Channel* object associated with the specified *RouterSide* object. The channel assigned to a *RouterSide* object is determined when you *add the router* to the *Routers* collection.
Availability	Local, full, and lightweight clients.
Syntax	*channelObject* = *object*.**Channel** **Element** — **Description** *channelObject* — The *Channel* object. *object* — The *RouterSide* object.
Data Type	*Channel* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Routers* object in the *ConstClassIds* constant: **11 lcaClassIdRouterSide** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

DetailInfo

Summary	Contains the *DetailInfo* object associated with the specified *RouterSide* object. The *DetailInfo* object contains an error log and communications status information for the *RouterSide* object. When you read this property from a *RouterSide*, OpenLNS will send a query to the device to obtain this information.
Availability	Local, full, and lightweight clients.
Syntax	*detailInfoObject* = *object*.*DetailInfo* **Element** — **Description** *detailInfoObject* — The *DetailInfo* object associated with the application device. *object* — The *RouterSide* object from which to get status information.
Data Type	*DetailInfo* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LastTestInfo

Summary	Contains the *TestInfo* object containing the results of the last time the *Test* method was called for this router side. The *AuxResultData* property indicates which properties of the *LastTestInfo* object contain useful information.
Availability	Local, full, and lightweight clients.
Syntax	*lastTestInfo* = *adObject.LastTestInfo* **Element** — **Description** *lastTestInfo* — The *TestInfo* object containing the last test results. *adObject* — The *RouterSide* object to be acted on
Data Type	*TestInfo* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NeuronId

Summary	Contains the Neuron ID associated with the router side. Neuron IDs are stored as 12-digit hexadecimal strings (for example, "a327ff27ba44").
Availability	Local, full, and lightweight clients.
Syntax	*neuronIdValue* = *object.NeuronId* **Element** — **Description** *neuronIdValue* — The *NeuronId* of the object. *object* — The *RouterSide* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

NodeId

Summary	Contains the node ID associated with a router side. The *NodeId* and *SubnetId* comprise the logical network address assigned to a *RouterSide* when the *Add* method is invoked on the *Router* that contains the *RouterSide* object. Each *RouterSide* is allocated a single node ID. **Note**: As of OpenLNS, you can write to this property.
Availability	Local, full, and lightweight clients.
Syntax	*nodeIdValue* = *object*.**NodeId**

	Element	**Description**
	nodeIdValue	The *NodeId* of the object.
	object	The *RouterSide* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PendingNeuronId

Summary	Contains the neuron ID that will be set when the router side is commissioned. If a router side is commissioned with one Neuron ID, then replaced, then the new Neuron ID is set, the *NeuronId* property will still show the old Neuron ID until the *Commission* method is called. This property allows the new Neuron ID to be read before commissioning.
Availability	Local, full, and lightweight clients.
Syntax	*neuronId* = *adObject*.**PendingNeuronId** **Element** — **Description** *neuronId* — The pending neuron ID. *adObject* — The *RouterSide* object to be acted on.

Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Priority

Summary	Determines the router side's priority assignment on its channel. An assignment of **0** indicates that the object is not assigned a priority slot. If the object is not assigned a priority slot, messages with priority will still use priority buffers, and will still be sent before messages without priority. In addition, messages with priority have the priority bit set, so that routers will send them out using priority buffers. If a router has a priority slot defined, the message will be forwarded on that slot.
Availability	Local, full, and lightweight clients.
Syntax	*priorityValue* = o*bject.Priority* **Element** — **Description** *priorityValue* — The priority value assigned to the object. The enumerated values that you can set for this property, which are stored in the *ConstLNSNodePriority* constant, are as follows: **127 lcaLNSNodePriorityMax** Represents the maximum number of priorty slots on any LonTalk channel. Do not set the *MaxPriority* property to a value greater than this. **255 lcaLNSNodePriorityAny** The Object Server will assign the *RouterSide* object the next available, or least used, priority slot on the channel. *object* — The *RouterSide* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ProgramId

Summary	Stores the program ID for the router side. Every LonMark compliant LONWORKS device uses a unique, 16-digit, hexadecimal standard program ID that uses

	the following format: FM:MM:MM:CC:CC:UU:TT:NN **Note**: You may not use colons when writing the program ID. For a device with a program ID of 80:00:01:01:28:80:04:02, you can write 8000010128800402 in this property. See the *Devices Interfaces* section in the *OpenLNS Programmer's Guide* for a description of the format used to display program IDs.
Availability	Local, full, and lightweight clients.
Syntax	*programIdValue* = *object*.**ProgramId** **Element** — **Description** *programIdValue* — The program ID value of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

State

Summary	Describes the state of a router side. You cannot set the state of a router side in the same transaction as an *Add*, *Commission*, or *Replace* method. You cannot set the state of a router side during the definition phase of the predefined components installation scenario. The state change may fail if the router side has not been completely updated due to a previous communication problem or if the router side has not been commissioned. If a failure occurs, use the *Commission* method to force an update of the router side. Router sides are placed offline while they are being configured or commissioned.
Availability	Local, full, and lightweight clients.
Syntax	*stateValue* = *object*.*State* **Element** — **Description** *stateValue* — The state of the router side (online or offline). The enumerated values for this property, which are stored in the *ConstDeviceStates* constant, are as follows: **Note:** The only two values that may be written to this property are **lcaStateCnfgOnline**(4) and **lcaStateSoftOffline**(12). All other

		properties are read-only. **2 lcaStateUncnfg** The application is loaded but the configuration is either not loaded, being reloaded, or deemed corrupted due to a configuration checksum error. A Neuron Chip also can make itself unconfigured by calling the Neuron C function **go_unconfigured()**. The device's service LED flashes at a one second rate in this state. **3 lcaStateNoApplUncnfg** No application is loaded yet, the application is in the process of being loaded, or the application has been deemed corrupted due to an application checksum error or signature inconsistency. The application does not run in this state. The device's service LED is steadily on in this state. **4 lcaStateCnfgOnline** Normal device state. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The device's service LED is off in this state. **6 lcaStateCnfgOffline** Application loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The device's service LED is off in this state. **12 lcaStateSoftOffline** The device has an application, is configured, and is soft-offline. It will go online when it is reset or when requested to go online. The device's service LED is off in this state. **140 lcaStateCnfgBypass** The application confirmed the offline request, but is still running (bypass

	mode). The device's service LED is off in this state. *object* — The *RouterSide* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

SubnetId

Summary	Contains the ID of the subnet. The *RouterSide* object's *SubnetId* property identifies the subnet the device is part of. This property can be used in conjunction with the *NodeId* property to uniquely identify a device. **Note**: As of OpenLNS, you can write to this property.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.*SubnetId* **Element** — **Description** *returnValue* — The subnet ID of the device. *object* — The *RouterSide* to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Subnets

Summary	Contains the *Subnets* collection object associated with the specified *RouterSide*. The *Subnets* collection contains subnets that are associated with *object*. For example, the *RouterSide* object's *Subnets* property returns a *Subnets* collection object corresponding to the subnet for the device's index 0 domain entry.
Availability	Local, full, and lightweight clients.
Syntax	*subnetCollection* = *object*.**Subnets** **Element** — **Description** *subnetCollection* — The returned *Subnets* collection *object* — The *RouterSide* object to be acted upon.
Data Type	*Subnets* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ServiceStatus

A *ServiceStatus* object contains the status of the current service request for the *System* object. This object contains information regarding whether a service is waiting for a resource such as a transaction, (if so, which resource and how many other services are ahead of this one) or if a service is proceeding. This object, unlike most others, can be accessed during an *OnSystemNssIdle* callback, which allows an OpenLNS client to check on the status of a service while it is blocked waiting for the method or property involved in the service to return.

The following table summarizes the *ServiceStatus* object.

Description	Contains the status of the current service request for the *System* object.
Added to API	LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *Parent* • *QueuePosition* • *ResourceType* • *Status*

Methods

The *ServiceStatus* object does not contain any methods.

Properties

The *ServiceStatus* object contains the following properties.

- *ClassId*
- *Parent*
- *QueuePosition*
- *ResourceType*
- *Status*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description**

	classIdValue	The object class of the object. The following value is defined for the *ServiceStatus* object in the *ConstClassIds* constant: **85 lcaClassIdServiceStatus**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.	
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.	
Syntax	*parentObject* = *object*.**Parent**	
	Element	**Description**
	parentObject	The object that is the parent of the specified *object*.
	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

QueuePosition

Summary	Indicates the numbed of transactions that must be started and completed before this service can be executed. If the *ServiceStatus* object's *Status* property is set to **lcaSrvcQueued**, this property will indicate the number of unstarted transactions that need to be completed before this one. For example, if this property is set to 0, the service will start as soon as the current transaction is completed. If the *ServiceStatus* object's *Status* property is set to a non-zero value, this property has no meaning.
Availability	Local, full, and lightweight clients.
Syntax	*queuePos* = *serviceStatusObject*.**QueuePosition** **Element** — **Description** *queuePos* — The queue position of this service. *serviceStatusObject* — The *ServiceStatus* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ResourceType

Summary	Indicates the resource for which the service is waiting. If the *ServiceStatus* object's *Status* property is set to **lcaSrvcQueued**, this property will indicate the number of unstarted transactions that need to be completed before this one. For example, if this property is set to 0, the service will start as soon as the current transaction is completed. If the *ServiceStatus* object's *Status* property is set to a non-zero value, this property has no meaning.
Availability	Local, full, and lightweight clients.
Syntax	*resType* = *serviceStatusObject*.**ResourceType** **Element** — **Description** *resType* — The enumerated value indicating the resource type. The possible values for this property, which are contained in the *ConstResourceType* constant, are as follows: **0 lcaResTransactions** Indicates that the service is waiting for one or more transactions to complete before it can execute. The *QueuePosition* property

		indicates the number of unstarted transactions that must be completed before the service can execute. **1 lcaResServicePin** Reserved for future use.
	serviceStatusObject	The *ServiceStatus* object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

Status

Summary	Indicates the status of the service that the *System* is currently executing.	
Availability	Local, full, and lightweight clients.	
Syntax	*statusValue = ssObject.Status*	
	Element	**Description**
	ssObject	The *ServiceStatus* object to be acted on. The possible values for this element, which are contained in the *ConstServiceStatus* constant, are as follows: **0 lcaSrvcInactive** There is no service currently being executed by this client. **1 lcaSrvcProgress** The service is currently being executed by this client. **2 lcaSrvcQueued** The service is currently waiting for transactions to complete before it can be executed. The number of transactions that must be started and completed before this service can be executed is contained in the *QueuePosition* property. See *Transactions* in the *OpenLNS Programmer's Guide* for more information. **3 lcaSrvcCommFailure**

		The service is unable to execute do to a communications failure.
	statusValue	The status of this service.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

SourceAddress

The *SourceAddress* object is used to indicate the source device of an update reported by the *OnMsgMonitorPointUpdateEvent* or *OnNvMonitorPointUpdateEvent* events. You should not use the source address to determine the data source of an update while performing runtime monitor and control operations is not recommended because it may impact network performance. Instead, your monitoring application should create unique *tag* values for each monitor point, and these values should be used to uniquely identify the source of the update. For more information on tag values, see the *Tag* property of the *ObjectServer*.

The following table summarizes the *SourceAddress* object.

Description	Indicates the source device that generated a network variable or message monitor point update.
Added to API	LNS Release 3.0.
Accessed Through	None.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *DomainId* • *NodeId* • *Parent* • *SubnetId*

Methods

The *SourceAddress* object does not contain any methods.

Properties

The *SourceAddress* object contains the following properties.

- *ClassId*
- *DomainId*
- *NodeId*
- *Parent*
- *SubnetId*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *SourceAddress* object in the *ConstClassIds* constant: **82 lcaClassIdSourceAddress** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

DomainId

Summary	Contains the domain ID of the device which generated the network variable or message monitor point update. The domain ID is stored as a string of hexadecimal digits. For example, a 3-byte domain ID would be represented as follows: "32a0cf". Domain IDs can be 1, 3, or 6 bytes in length. If the network variable of message monitor point update came from a device using Neuron ID addressing that is on a different domain than the OpenLNS database, this property contains an empty string.
Availability	Local and full clients.
Syntax	*domainIdValue* = *srcAddrObject*.**DomainId** **Element** — **Description** *domainIdValue* — The domain ID for the subnet. *srcAddrObject* — The *SourceAddress* object to be acted on.
Data Type	String.
Read/Write	Read only.

Added to API	LNS Release 3.0.

NodeId

Summary	Contains the node ID of the device that generated the network variable or message monitor point update.
Availability	Local, full, and lightweight clients.
Syntax	*nodeIdValue* = *object*.**NodeId** **Element** — **Description** *nodeIdValue* — The *NodeId* of the object. *object* — The *SourceAddress* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

SubnetId

Summary	Contains the subnet ID of the device that generated the network variable or message monitor point update. A device's *SubnetId* property identifies the subnet on which the device resides. This property can be used in conjunction with the *NodeId* property to uniquely identify a device.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.*SubnetId* **Element** — **Description** *returnValue* — The subnet ID of the device. *object* — The *SourceAddress* to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Subnet

The *Subnet* object represents a single instance of a subnet. Subnets are the second component of the three-component LonTalk addressing hierarchy of domain/subnet/node ID. The subnet address is the level at which routers decide whether to forward a packet; therefore, the same subnet cannot appear on both sides of a configured or learning router.

Subnets are typically added automatically when routers or devices are added. In most cases there is no reason for your application to manually allocate subnets. Subnets can be explicitly added using the *Add* method of the *System* object's *Subnets* collection. This allows an OpenLNS application to force a given set of devices onto the same subnet by specifying that subnet when adding the device, or by subsequently moving the device to another subnet with the *MoveEx* method. This will allow your application to take advantage of subnet broadcast messaging.

The following table summarizes the *Subnet* object.

Description	Represents a single instance of a subnet.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Subnets* collection object.
Default Property	*Name*.
Methods	None.
Properties	• *BitmapFilePath* • *ClassId* • *Description* • *DomainId* • *Extensions* • *IconFilePath* • *Name* • *Parent* • *SubnetId*

Methods

The *Subnet* object does not contain any methods.

Properties

The *Subnet* object contains the following properties.

- *BitmapFilePath*
- *ClassId*
- *Description*
- *DomainId*
- *Extensions*
- *IconFilePath*
- *Name*
- *Parent*
- *SubnetId*

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.
Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** — **Description** *bmpFilePath* — The bitmap path and file name. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to

	Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Subnet* object in the *ConstClassIds* constant: **16 lcaClassIdSubnet** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Description

Summary	Stores description information about the *Subnet* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *Subnet* object. *object* The *Subnet* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

DomainId

Summary	Contains the domain ID for a subnet. The domain ID is stored as a string of hexadecimal digits. For example, a 3-byte domain ID would be represented as follows: "32a0cf". Domain IDs can be 1, 3, or 6 bytes in length.
Availability	Local, full, and lightweight clients.
Syntax	*domainIdValue* = *subnetObject*.**DomainId** **Element** **Description** *domainIdValue* The domain ID for the subnet.

	subnetObject The *Subnet* object.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *Subnet*. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.*Extensions* **Element** — **Description** *extensionsColl* — The *Extensions* collection object. *object* — The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (*.**ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** — **Description** *IconFilePathFileName* — Icon file and path name *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors

	• Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.

Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

SubnetId

Summary	Contains the ID of the subnet.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.*SubnetId* **Element** — **Description** *returnValue* — The subnet ID of the device. *object* — The *Subnet* to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Subnets

The *Subnets* object represents a collection of *Subnet* objects. Subnets are typically added automatically when routers or devices are added. In most cases there is no reason for your application to manually allocate subnets. Subnets can be explicitly added using the *Add* method of the *System* object's *Subnets* collection. This allows an LCA application to force a given set of devices onto the same subnet to be able to take advantage of subnet broadcast messaging.

- The *Subnets* collection accessed through the *AppDevice* or *RouterSide* objects always returns a collection containing the single subnet associated with the device's primary domain.
- The *Subnets* collection accessed through the *System* object represents all of the subnets in a system.

The following table summarizes the *Subnets* object.

Description	A collection of *Subnet* objects.
Added to API	LNS Release 3.0.
Accessed Through	*AppDevice* object. *RouterSide* object. *System* object.
Default Property	*Item*
Methods	• *Add* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *Subnets* object contains the following methods.

- *Add*
- *Remove*

Add

Summary	Defines a new *Subnet* object. Subnets are typically added automatically when routers or devices are added. In most cases there is no reason for your application to manually allocate subnets. Subnets can be explicitly added using this method. You can then force a given set of devices onto the same subnet with the *MoveEx* method take advantage of subnet broadcast messaging.
Availability	Local, full, and lightweight clients.
Syntax	*subnetObject* = *subnetsColl*.**Add**(*subnetName*, *subnetId*) **Element** — **Description** *subnetObject* — The newly defined subnet object. *subnetsColl* — The *Subnets* collection object. *subnetName* — A String containing the name of the subnet. *subnetId* — An Integer value indicating the subnet ID. This element may be a value from 0–255. A value of **0** indicates that the Object Server should assign the subnet ID.
Added to API	Prior to LNS Release 3.0.

Remove

Summary	Removes a subnet from the system. A subnet may only be removed if there are no devices assigned to it.
Availability	Local, full, and lightweight clients.
Syntax	*subnetsColl*.**Remove** *indexName* **Element** — **Description** *subnetsColl* — The *Subnets* collection object containing the subnet to be removed. *indexName* — A Long value specifying the collection index of the *Subnet* object to be removed, or a String value specifying the name of the *Subnet* object to be removed.

Added to API	Prior to LNS Release 3.0.

Properties

The *Subnets* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Subnets* object in the *ConstClassIds* constant: **17 lcaClassIdSubnets** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer.

	object The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *Subnet* object from a *Subnets* collection. You can retrieve a *Subnet* object from its *Subnets* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *Subnet* object in *Subnets* collections with the *Name* property by passing the *Subnet* object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *Subnet* object retrieved from the collection. *collObject* — The *Subnets* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *Subnet* object to be retrieved. *stringExpression* — A string type specifying the name of the *Subnet* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent

	clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

Subsystem

The *Subsystem* object represents a logical grouping of devices and routers. Subsystems may be hierarchical. Each user-defined subsystem may itself contain additional user-defined subsystems.

The *System* object's *Subsystems* collection contains the top-level subsystems in the user's subsystem hierarchy plus two pre-defined subsystems: the ALL and the Discovered subsystem. These predefined *Subsystem* objects are described as follows:

- The **ALL** subsystem is always given index **1** within the *System* object's *Subsystems* collection. This subsystem contains all objects in the network database, including the subsystems in the system that are managed by the LNS client applications. However, this also includes transient objects that OpenLNS uses that should not be accessed by client applications. In general, you should not use the **ALL** subsystem, and you should not iterate through this subsystem to get a list of devices or routers on a network.

  The **ALL** subsystem, however, may be useful if you are not sure to which subsystem a device or router belongs, but you know the handle or Neuron ID of the device or router. In that case, you can locate the device by accessing *AppDevices* or *Routers* collection of the **ALL** subsystem, and invoking the *ItemByHandle* or *ItemByNeuronId* methods.

  A newly discovered uninstalled router (by service pin, or device discovery) will not appear in the **ALL** subsystem's *RouterDevices* collection until the **Discovered.Uninstalled.RouterDevices** collection is accessed.

- The **Discovered** subsystem is always given index **2** in the *System* object's *Subsystems* collection. The *AppDevices* and *RouterDevices* properties in the **Discovered** subsystem are empty collections; however, this pre-defined subsystem contains two special-purpose subsystems: the **Installed** and **Uninstalled** subsystems. These two special-purpose subsystems are described as follows:

  - The **Discovered.Installed** subsystem is always at Index 1 in the **Discovered** subsystem. It contains all devices that were not explicitly added to a subsystem by an OpenLNS client. For example, all NSDs will initially be in this subsystem because they are created implicitly by OpenLNS. If you use the *Move* or *MoveEx* method to move an NSD or application device in the **Disovered.Installed** subsystem to a user subsystem manually, or if you use the *AddReference* method to add a reference to a device in the **Discovered.Uninstalled** subsystem to a user subsystem, the device will be removed from the **Discovered.Installed** subsystem. Devices discovered by network recovery operations are also stored in this subsystem, until they are moved by an application to a user subsystem.

  - The **Discovered.Uninstalled** subsystem is always at Index 2 in the **Discovered** subsystem. It contains unconfigured devices that have been automatically discovered on the network or devices that have been discovered by the *System* object's *DiscoverDevices* method.

    Write access to the application devices and routers contained within these subsystems is limited. Devices may not be added to or deleted from these subsystems; however, they may be moved to other subsystems, effectively installing them using the *AddReference* method of the *AppDevices* or *Routers* collections. You can rename the application devices and routers stored in these subsystems by writing to their *Name* property, but you cannot commission them or modify their Neuron IDs until you move them to another subsystem.

Note: The name of a top-level *Subsystem* can not be the same as the *System* object's name. See the *Path* property for more information.

The following table summarizes the *Subsystem* object.

Description	Represents a logical grouping of devices and routers.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Subsystems* property.
Default Property	*Name*.
Methods	None.
Properties	• *AppDevices* • *BitmapFilePath* • *ClassId* • *Description* • *Extensions* • *Handle* • *IconFilePath* • *Name* • *Parent* • *Path* • *RouterDevices* • *Subsystems*

Methods

The *Subsystem* object does not contain any methods.

Properties

The *Subsystem* object contains the following properties.

- *AppDevices*
- *BitmapFilePath*
- *ClassId*
- *Description*
- *Extensions*
- *Handle*
- *IconFilePath*
- *Name*
- *Parent*
- *Path*
- *RouterDevices*
- *Subsystems*

AppDevices

Summary	Contains the *AppDevices* collection object associated with the specified *Subsystem* object. The *AppDevices* property represents all the devices on the channel.
Availability	Local, full, and lightweight clients.

Syntax	*appDevicesCollection* = *object*.**AppDevices** **Element** **Description** *appDevicesCollection* The *AppDevices* collection returned. *object* The *Subsystem* object to be acted on.
Data Type	AppDevices collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

BitmapFilePath

Summary	Specify the path and file name of a bitmap (*.**BMP** file) representation of the object. The bitmap files are used to store object images which may be accessed by a director level LNS component application. A bitmap may be of any size, although the recommended dimensions are 40x80 pixels. See the *IconFilePath* property for related information.
Availability	Local clients.
Syntax	*bmpFilePath* = *object*.**BitmapFilePath** **Element** **Description** *bmpFilePath* The bitmap path and file name. *object* The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyBMPs\Object.BMP**).
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the

	Subsystem object in the *ConstClassIds* constant: **5 lcaClassIdSubsystem** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Description

Summary	Stores description information about the *Subsystem* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *Subsystem* object. *object* The *Subsystem* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *Subsystem*. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object*.*Extensions* **Element** **Description** *extensionsColl* The *Extensions* collection object. *object* The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.

Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *Subsystem* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property. Some OpenLNS Objects are tracked internally by the OpenLNS Server using a unique handle. You can fetch objects using the *ItemByHandle* method.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** — **Description** *returnValue* — The NSS handle of the object. *object* — The object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

IconFilePath

Summary	Specifies the path and file name of an icon (*.**ICO** file) representation of the object.
Availability	Local clients.
Syntax	*IconFilePathFileName* = *object*.**IconFilePath** **Element** — **Description** *IconFilePathFileName* — Icon file and path name *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write. If you write to this property, you must specify the bitmap's full path and file name (for example, **C:\MyICOs\Object.ICO**). The icon file should contain the following representations: • Standard (32x32 pixels) with 256 colors • Small (16x16) with 16 colors • Monochrome (32x32) • Large (48x48) with 256 colors
Added to API	Prior to LNS Release 3.0.

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Path

Summary	Contains the full system and subsystem pathname for this subsystem. Pathnames may be used in place of names when fetching *Subsystem* objects from the *System*'s *Subsystems* collection via the *Item* property. The pathname for a subsystem is the system name and the subsystem hierarchy leading to that subsystem, with all names separated by periods. The system name is optional. An example pathname is **MySystem.Discovered.Installed**. None of the top-level subsystems can have the same name as the system because the resulting subsystem path name would be ambiguous (for example, if a top-level subsystem in the **My System** system was named **My System**, a pathname of "**My System**" could represent either the system name or a top-level subsystem name). If this pathname is to be passed remotely, it may be up to 170 characters in length. If this pathname is only to be used locally, there is no defined limit. Pathnames greater that 170 characters must not be used if remote OpenLNS clients will be used that need to access the subsystems with long path names.
Availability	Local, full, and lightweight clients.
Syntax	*subsystemPath* = *subsystemObject*.**Path** **Element** — **Description** *subsystemPath* — The full pathname of the subsystem. *subsystemObject* — The *Subsystem* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

RouterDevices

Summary	Returns the *Routers* collection for the specified *Subsystem*. This collection contains all the *Router* objects installed in the specified *Subsystem*. **Note**: A newly discovered uninstalled router (by service pin, or device discovery) will not appear in the **ALL** subsystem's *RouterDevices* collection until the **Discovered.Uninstalled.RouterDevices** collection is accessed.
Availability	Local, full, and lightweight clients.
Syntax	*routersCollection* = *object*.**RouterDevices**

	Element **Description** *object* The *Subsystem* object to be acted on. *routersCollection* The *Routers* collection returned.
Data Type	*Routers* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Subsystems

Summary	Contains the *Subsystems* collection object associated with the specified *Subsystem* object. A *Subsystems* collection accessed through a *Subsystem* object can be used to represent and define subsystem hierarchy. For example, you could define a subsystem representing a building. The building subsystem could consist of separate subsystems for each floor, and each floor subsystem could contain a set of subsystems for each room on the floor. The room subsystem could then contain separate subsystems for HVAC, security, and lighting subsystems. Defining subsystem hierarchies in this way should allow you to keep the application devices and routers in your network organized by location.
Availability	Local, full, and lightweight clients.
Syntax	*subsystemCollection = object.Subsystems* **Element** **Description** *subsystemCollection* The *Subsystems* collection associated with the object. *object* The *Subsystem* object.
Data Type	*Subsystems* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Subsystems

The *Subsystems* object represents a collection of *Subsystem* objects. A *Subsystems* collection accessed through an *AppDevice* or *Router* object represents all of the subsystems which contain that *AppDevice* or *Router* object.

A *Subsystems* collection accessed through a *Subsystem* object can be used to represent and define subsystem hierarchy. For example, you could define a subsystem representing a building. The building subsystem could consist of separate subsystems for each floor, and each floor subsystem could contain a set of subsystems for each room on the floor. The room subsystem could then contain separate subsystems for HVAC, security, and lighting subsystems. Defining subsystem hierarchies in this way should allow you to keep the application devices and routers in your network organized by location.

The *Subsystems* collection accessed through the **ALL** subsystem contains all objects in the network database, including the subsystems in the system that are managed by the LNS client applications. This includes transient objects that OpenLNS uses that should not be accessed by client applications. In general, you should not use the **ALL** subsystem.

The following table summarizes the *Subsystems* object.

Description	A collection of *Subsystem* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*AppDevice* object. *Router* object. *Subsystem* object. *System* object.
Default Property	*Item*
Methods	• *Add* • *ItemByHandle* • *Remove*
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *Subsystems* object contains the following methods.

- *Add*
- *ItemByHandle*
- *Remove*

Add

Summary	Defines a new *Subsystem* object. A subsystem is an object that contains other subsystems, application devices, and routers. A subsystem is an object that contains other subsystems, application devices, and routers. *Subsystems* can be used to organize devices just as directories can be used to organize files. For example, each subsystem could represent groupings of devices in a room, or on a single floor of a building. For more detailed information about subsystems and how they fit into the OpenLNS Object Hierarchy, see the *OpenLNS Object Server Hierarchy* section in the *OpenLNS Programmer's Guide.*
Availability	Local, full, and lightweight clients.

Syntax	*subsystemObject* = *subsystemsColl*.**Add**(*subsystemName*) **Element** **Description** *subsystemObject* The newly defined *Subsystem* object. *subsystemsColl* The *Subsystems* collection object. *subsystemName* A String containing the name of the *Subsystem*.
Added to API	Prior to LNS Release 3.0.

ItemByHandle

Summary	Retrieves a *Subsystem* object, specified by its handle, from a *Subsystems* collection.
Availability	Local, full, and lightweight clients.
Syntax	*object* = *coll*.*ItemByHandle*(*handle*) **Element** **Description** *object* The *Subsystem* object retrieved from the *Subsystems* collection. *coll* The *Subsystems* collection object. *handle* A Long value specifying the handle of the *Subsystem* object to be retrieved.
Added to API	LNS Release 3.0.

Remove

Summary	Removes a *Subsystem* object from the *Subsystems* collection.
Availability	Local, full, and lightweight clients.
Syntax	*objectColl*.**Remove** *indexName* **Element** **Description** *objectColl* The *Subsystems* collection object containing the *Subsystem* object to be removed. *indexName* A Long value specifying the collection index of the *Subsystem* object to be removed, or a String value specifying the name of the *Subsystem* object to be removed.
Added to API	Prior to LNS Release 3.0.

Properties

The *Subsystems* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassID

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Subsystems* object in the *ConstClassIds* constant: **6 lcaClassIdSubsystems** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.

Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *Subsystem* object from a *Subsystems* collection. You can retrieve a *Subsystem* object from its *Subsystems* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *Subsystem* object in *Subsystems* collections with the *Name* property by passing the *Subsystem* object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The object retrieved from the collection. *collObject* — The collection object to be acted on. *index* — A Long type specifying the ordinal index of the object to retrieve. *stringExpression* — A string type specifying the name of the object to retrieve.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent**

	Element **Description** *parentObject* The object that is the parent of the specified *object*. *object* Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** **Description** *retrievedObject* An expression that evaluates to the object retrieved. *collObject* An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

System

A *System* object represents a system managed by a single OpenLNS Server. There is one *System* per *Network* object. The following table summarizes the *System* object.

Description	A system managed by a single OpenLNS Server.

Added to API	Prior to LNS Release 3.0.
Accessed Through	*Systems* collection object.
Default Property	*Name*.
Methods	• *BeginAttachmentEvent* • *BeginChangeEvent* • *BeginCommissionEvent* • *BeginLicenseEvent* • *BeginLonMarkObjectStatusChangeEvent* • *BeginMissedEvent* • *BeginNodeConnChangeEvent* • *BeginNodeIntfChangeEvent* • *BeginNssIdleEvent* • *BeginServicePinEvent* • *BeginSession* • *BeginSystemMgmtModeChangeEvent* • *CancelTransaction* • *Close* • *CommittTransaction* • *DeconfigNetwork* • *DiscoverDevices* • *DoEventSync* • *DsRestoreOptions* • *DsSaveOptions* • *EndAttachmentEvent* • *EndChangeEvent* • *EndCommissionEvent* • *EndLonMarkObjectStatusChangeEvent* • *EndMissedEvent* • *EndNodeConnChangeEvent* • *EndNodeIntfChangeEvent* • *EndNssIdleEventEnd* • *EndServicePinEvent* • *EndSession* • *EndSystemMgmtModeChangeEvent* • *ExtensionByHandle* • *GetPermission* • *GetProgramId* • *Open* • *PrepareToRecoverFromNetwork* • *RecoverFromNetwork* • *RetryUpdates* • *RestoreLicense* • *SetEventSyncMode* • *StartTransaction* • *WinkByNeuronId*
Properties	• *Accounts* • *ActivationLicense* • *ApplicationHandle* • *ApplicationName* • *AuthenticationKey*

	• *ClassId* • *ClientId* • *CommissionedDeviceCount* • *ComponentApps* • *Connections* • *CurrentAccount* • *CurrentDeviceCount* • *CustomerId* • *DebugTraceFlag* • *Description* • *DiscoveryInterval* • *DiscoveryLimitedFlag* • *DomainId* • *DsPollInterval* • *DsPriority* • *DsRepeatTimer* • *DsRetries* • *DsRetryCount* • *DsTxTimer* • *Extensions* • *FileTransfer* • *Handle* • *HostTimer* • *ImportDirectory* • *InstallOptions* • *IsOpen* • *LastError* • *LaunchLcaServerFlag* • *LdrfCatalogPath* • *LdrfLanguages* • *MgmtMode* • *Name* • *NetworkResources* • *NetworkServiceDevice* • *NssDbVersion* • *Parent* • *PermissionString* • *PingIntervals* • *RecoveryStatus* • *RegisterServicePin* • *RemoteChannel* • *RepeatTimer* • *ResourceLanguageId* • *RetryCount* • *SecurityLevel* • *ServiceStatus* • *State* • *Subnets* • *Subsystems* • *TemplateLibrary* • *TxTimer* • *UncommissionedDeviceCount*

	• *UninstalledDeviceCount* • *UpdateInterval*

Methods

The *System* object contains the following methods:

- *BeginAttachmentEvent*
- *BeginChangeEvent*
- *BeginCommissionEvent*
- *BeginLicenseEvent*
- *BeginLonMarkObjectStatusChangeEvent*
- *BeginMissedEvent*
- *BeginNodeConnChangeEvent*
- *BeginNodeIntfChangeEvent*
- *BeginNssIdleEvent*
- *BeginServicePinEvent*
- *BeginSession*
- *BeginSystemMgmtModeChangeEvent*
- *CancelTransaction*
- *Close*
- *CommittTransaction*
- *DeconfigNetwork*
- *DiscoverDevices*
- *DoEventSync*
- *DsRestoreOptions*
- *DsSaveOptions*
- *EndAttachmentEvent*
- *EndChangeEvent*
- *EndCommissionEvent*
- *EndLonMarkObjectStatusChangeEvent*
- *EndMissedEvent*
- *EndNodeConnChangeEvent*
- *EndNodeIntfChangeEvent*
- *EndNssIdleEventEnd*
- *EndServicePinEvent*
- *EndSession*
- *EndSystemMgmtModeChangeEvent*
- *ExtensionByHandle*
- *GetPermission*
- *GetProgramId*
- *Open*
- *PrepareToRecoverFromNetwork*
- *RecoverFromNetwork*
- *RetryUpdates*
- *RestoreLicense*
- *SetEventSyncMode*
- *StartTransaction*
- *WinkByNeuronId*

BeginAttachmentEvent

Summary	Enables the *OnAttachmentEvent* for a system for a given

	object type. To begin events for device and router attachments, you must call this method twice: once with the **lcaAppDeviceEvent** parameter, and once with the **lcaRouter** parameter.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.BeginAttachmentEvent objectType* **Element** — **Description** *systemObject* — The *System* object to be acted on. *objectType* — An integer determining the object type for which to begin attachment events. The possible values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**
Added to API	Prior to LNS Release 3.0.

BeginChangeEvent

Summary	Enables the *OnChangeEvent* for a system for a given object type. You must call this method once for every type of object for which change events are to be received.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.***BeginChangeEvent** *objectType* **Element** — **Description** *systemObject* — The *System* object to be acted on. *objectType* — An integer determining the object type for which to begin change events. The possible values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**
Added to API	Prior to LNS Release 3.0.

BeginCommissionEvent

Summary	Enables the *OnCommission* event for a system for a given

	object type.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*.**BeginCommissionEvent** *objectType* **Element** — **Description** *systemObject* — The *System* object to be acted on. *objectType* — An integer determining the object type for which commission events will be enabled. The possible values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**
Added to API	Prior to LNS Release 3.0.

BeginLonMarkObjectStatusChangeEvent

Summary	Registers your application for the *OnLonMarkObjectStatusChange* event. The *OnLonMarkObjectStatusChange* event will then be fired each time an OpenLNS client application changes the status of a *LonMarkObject* on the *System* . OpenLNS applications are required to respond to license expiration events as specified in the OpenLNS license agreement. You can use this method to begin monitoring for these events.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*. **BeginLonMarkObjectStatusChangeEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

BeginMissedEvent

Summary	Enables the *OnMissedEvent* for a system. You should call this method at initialization if missed events will be used. In addition, to enable missed events, this method also causes a ping event to be sent to the NSI every 60 to 90 seconds to ensure that it is still there to receive events.
Availability	Local, full, and lightweight clients.

Syntax	*systemObject.BeginMissedEvent* **Element** **Description** *systemObject* The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

BeginNodeConnChangeEvent

Summary	Enables the *OnNodeConnChangeEvent* for a system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.BeginNodeConnChangeEvent* **Element** **Description** *systemObject* The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

BeginNodeIntfChangeEvent

Summary	Enables the *OnNodeIntfChange* event for a system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.* **BeginNodeIntfChangeEvent** **Element** **Description** *systemObject* The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

BeginNssIdleEvent

Summary	Enables the *OnSystemNssIdle* event for a system. You must call the *EndNssIdleEvent* method before doing anything that may render your handler incapable of responding to the *OnSystemNssIdle* event. The *OnSystemNssIdle* event will be fired while your application is waiting for lengthy network operations to complete. Instances of this event will be returned synchronously, and if your application does not handle the event in a timely manner, your application may hang.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.* **BeginNssIdleEvent** *maxIntervalTime* **Element** **Description** *systemObject* The *System* object to be acted on. *maxIntervalTime* A Long value specifying the maximum time between idle events in milliseconds. The valid range for

	this element is 0–65,535.
Added to API	Prior to LNS Release 3.0.

BeginServicePinEvent

Summary	Enables the *OnSystemServicePin* event for a system. An event tag is returned from the method. Multiple service pin events can be enabled at one time, each having a different tag. When the service pin event is fired, the event tag is returned as one of the parameters.
Availability	Local, full, and lightweight clients.
Syntax	*eventTag* = *systemObject*.**BeginServicePinEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on. *eventTag* — An Integer representing an allocated event tag.
Added to API	Prior to LNS Release 3.0.

BeginSession

Summary	Begins a session for methods and properties that must be grouped to avoid intermediate error conditions. Currently, this only applies to operations that affect the physical topology of a network. All methods invoked and properties written between the *BeginSession* and *EndSession* methods are considered atomic. This method allows your application to create connections more efficiently, and avoid failure scenarios that can occur when devices or routers are moved or changed one-by-one. This method differs from the *StartTransaction* method in that services in a transaction include validation and update the OpenLNS database as they are invoked, although this update will not be committed until the transaction is committed. Sessions allow methods to be invoked and properties to be written that would individually cause an error. For example, consider moving a large number of devices with subnet broadcast connections and unacknowledged repeat service from one channel to another. If only some of these devices are moved, the OpenLNS Object Server would detect that not all of the devices are on the same subnet and the move would fail (since unacknowledged repeat service for domain wide broadcast is not allowed). By grouping the moves in a session the devices are not actually connected and

	validated until the session is ended. The only network operations you can perform within sessions are those related to changes in the physical topology of your network. These operations include moving devices and routers, adding and removing routers, and setting the class of routers. As a result, the methods you can use within a session include the following: *PreMove*, *MoveEx*, *PostMove*, *Add*, and *Remove*. You can also write to the *Class* property of a router. If you attempt to access any other properties, or invoke any other methods, within a session, then the **NS, #294 lcaErrNsNotAllowedInASession** exception will be thrown. A given client can have at most one session in progress at a time. A session must be part of an explicit transaction. A transaction may contain more than one session. Changes made in a session will not be committed until the transaction that contains that session is committed.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.BeginSession sessionClass* **Element** — **Description** *systemObject* — The *System* object to be acted on. *sessionClass* — The session class. Currently, the only valid value is **lcaSessionMove**, which is contained in the *ConstSessionClass* constant.
Added to API	Prior to LNS Release 3.0.

BeginSystemMgmtModeChangeEvent

Summary	Registers your application for the *OnSystemMgmtModeChangeEvent*. The *OnSystemMgmtModeChangeEvent* will then be fired each time the system's management mode changes. You can disable the *OnSystemMgmtModeChangeEvent* by invoking the *EndSystemMgmtModeChangeEvent* method. You can use the *MgmtMode* property to read or write to the system management mode.
Availability	Local, full, and lightweight clients.
Syntax	*system*.**BeginSystemMgmtModeChangeEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

CancelTransaction

Summary	Cancels an active transaction. When a transaction is canceled, either explicitly because of a reset, all network and database updates made since the call to the *StartTransaction* method are reversed. Once a transaction has been committed, it can no longer be cancelled. If you use the *CancelTransaction* method to end a transaction that was initiated as part of a new application program download and the download has already started, the device receiving the program download will be left in an applicationless state. However, this is not true if the transaction was canceled before the transaction actually started the download process. The *CancelTransaction* method may return an "update type error" indicating that the cancel has successfully reversed the modifications to the database, but has been unable to update one or more application devices. This will be an error in the NS error range, but will be without the **lcaErrRangeNsStart (1000)** value. To compare the update error types returned by the *CancelTransaction* method to the *constant error values* used in other OpenLNS operations, you will need to add the **lcaErrRangeNsStart (1000)** value to the returned error value. For example, consider a case where the *CancelTransaction* method returns error **63**. To compare the returned exception to the constant error values for the *NS error range*, you would need to add **lcaErrRangeNsStart (1000)** to the returned value. In this case, the returned error maps to the *lcaErrNsNothingToCancel* exception. Unlike most other methods, the *CancelTransaction* method may be called in an *OnSystemNssIdleSystem* event callback, which allows you to cancel the current method or property, and the transaction explicitly or implicitly containing it.
Availability	Local, full, and lightweight clients.
Syntax	*returnCode* = *systemObject*.**CancelTransaction** **Element** — **Description** *returnCode* — An Integer value which is assigned the return status code. *systemObject* — The *System* object whose transaction will be canceled.
Added to API	Prior to LNS Release 3.0.

Close

Summary	Closes the system, ends events, shuts down the NSS, and detaches the NSI from the network (if not opened in

	engineered system mode). Closing the system does not turn off monitor set monitoring and control. Monitor set monitoring and control continues until the *Network* object calls its *Close* method.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.Close* **Element** **Description** *systemObject* The *System* object to be closed.
Added to API	Prior to LNS Release 3.0.

CommittTransaction

Summary	Commits the current transaction. All database updates made since the transaction started will be marked as complete when you call this method. If the *MgmtMode* property is set to **lcaMgmtModePropagateConfigUpdates (0)** when you call this method, all database updates will be propogated to the devices on the network as part of the commit process. See the *CancelTransaction* method for information on canceling a transaction. If there is no active transaction, an error is returned. The *StartTransaction* method will fail and return an exception if any operation that modified the database failed during the transaction. If an "update type error" occurs, then the committed transaction has successfully modified the database, but has been unable to update one or more application devices. An "update type error" will cause a *Network* Service exception in the range 4030-4089 to be thrown. For more information on Network Service exceptions, see *Network Service Errors* in Chapter 4.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.StartTransaction* **Element** **Description** *systemObject* The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

DeconfigNetwork

Summary	Deconfigures the devices and routers associated with a system in a group of user-specified domains. This method is only available on Local client applications. It is typically used to reset devices in a development environment to a common state, or to prepare devices in a network for new installations that utilize the background

	discovery process (see the *DiscoveryInterval* property). Devices do not have to be installed by the OpenLNS Server to be deconfigured; however the following must be true: • The devices' domain IDs must match the domain IDs referenced by the *domainTypes* or *explicitDomain* elements passed to this method. • The devices must be in the configured online or soft offline state. • The devices must not be authenticated. The method's parameters specify the domains to deconfigure and the types of special-purpose devices to exclude. The possible uses of this method include: • To re-use devices from another network. You can invoke this service to ensure that devices are deconfigured, so that they do not interfere with traffic on the new network. In this case, you must specify the domain ID of the devices to deconfigure. Deconfiguring the devices also ensures that the OpenLNS Server can discover and query them using background discovery. • To deconfigure all devices. Before commissioning a new network using the predefined components installation scenario, you can invoke this method to deconfigure all devices. This ensures that after commissioning, any *AppDevice* object that was not defined with the *Add* method is unconfigured. • To refresh the entire network installation. You can invoke this method to deconfigure all devices, and then reload the network images for installed devices by invoking the *Commission* method, once for each device. This method does not use or affect device configuration information in the database. As a result, if devices are specified to be excluded from deconfiguration, the OpenLNS Server must individually discover and deconfigure each device. This process is very time consuming. Additionally, when deconfiguring a multi-channel network, if any device type is excluded, routers must also be excluded (using **lcaExcludedDevTypeRouters**). Otherwise, routers will be deconfigured along with the other devices, isolating remote channels and preventing all devices on those channels from being deconfigured reliably. When no devices are excluded (using **lcaExcludedDevTypeNone**), the OpenLNS Server deconfigures the devices using domain-wide broadcast messaging. All devices configured on the specified domain that can receive the broadcast will be deconfigured. Devices residing on channels other than the OpenLNS Server's will only be deconfigured if the intervening routers are also configured on the specified domain. When the domain

<table>
<tr><td></td><td>broadcast is used, all deconfigured devices will log a configuration checksum error (CNFG_CS_ERROR = 153) in their Neuron error log (see the DetailInfo object).

This method does not deconfigure the OpenLNS Server's NetworkServiceDevice. In fact, the OpenLNS Server is not required to be configured on the domain whose devices will be deconfigured. During execution of the DeconfigNetwork method, the OpenLNS Server's local NSI will be configured on the specified domain. When execution completes, the NSI is restored to its original domain.

Remote NSIs will be deconfigured, provided the NSI device type is not specified as an excluded device types (using lcaExcludedDevTypeNsMgrs). If a remote client's NSI is deconfigured, the client application must be shut down and restarted to regain access to the network.</td></tr>
<tr><td>Availability</td><td>Local clients.</td></tr>
<tr><td>Syntax</td><td>sysObject.DeconfigNetwork excludedDeviceTypes, domainTypes, explicitDomain

Element — Description

sysObject — The System object to be acted on.

excludedDeviceTypes — An Integer value indicating the devices to be excluded from deconfiguration.

The valid values for this element, which are contained in the ConstExcludedDevTypes constant, are as follows:

0 lcaExcludedDevTypeNone

All device types will be deconfigured.

1 lcaExcludedDevTypeLbMgrs

Devices with the <code>MICRO_SM</code> program ID (LonBuilder Network Manager device types), will not be deconfigured. Specify this program ID when your network includes the LonBuilder tool.

2 lcaExcludedDevTypeNsMgrs

Devices containing an OpenLNS Server will not be deconfigured. This option is recommended.

4 lcaExcludedDevTypeRouters

Routers will not be deconfigured.</td></tr>
</table>

		255 lcaExcludedDevTypeAll None of the above devices will be deconfigured. You can logically OR these constants to combine their effects.
	domainTypes	An Integer value indicating the types of domains to be deconfigured. The valid values for this element, which are contained in the *ConstDeconfigDomainTypes* constant, are as follows: **1 lcaDeconfigDomainCommon** Reserved for future use. **2 lcaDeconfigDomainAppl** Use the *DomainId* property of the *System* object to execute this method. Deconfigure all devices with that domain ID. **4 lcaDeconfigDomainExplicit** Use the specified domain to execute this method. You can logically OR these constants to combine their effects.
	explicitDomain	A String specifying a domain ID to use if the *domainTypes* element specifies the explicit domain option (see below). This allows a domain that is not part of the OpenLNS database to be used by this method
Added to API	Prior to LNS Release 3.0.	

DiscoverDevices

Summary	Discovers all configured devices on a specified domain, as well as all unconfigured devices on the network, and places them in the *Discovered.Uninstalled* subsystem. This method may be used to find packages of pre-installed nodes, or to discover nodes that were previously installed by another network management tool. This discovery operation overrides the system's default discovery operation, which only discovers unconfigured nodes. For more information on the system's default discovery operation, see the *DiscoveryInterval* property.

	If a host-based device whose application is not running is installed on the network, OpenLNS may or may not be able to discover it. Whether OpenLNS can discover and register the device depends on how much of the device's LonTalk protocol stack is running at the time of the discovery. Discovering a device involves first the actual discovery of the device and then the registration of that device in the database. The discovery service is always performed in the foreground and the registration service may be performed either as a background or foreground task. Select the type of operation by setting the *backgroundReg* parameter to True or False, as desired. If *backgroundReg* is set to False, devices are registered in the foreground as part of the discovery transaction, so the method will not return until discovery is completed or terminated. When the method completes, it returns the number of discovered devices in the *numDevicesDiscovered* parameter, and the newly registered devices may be accessed in the *Discovered.Uninstalled* subsystem. If an error occurs while registering a device, the method will return when one of the following conditions is met: all discovered devices have been registered, or 50 devices have been registered. If *backgroundReg* is set to True, discovered devices are registered in separate background transactions. In this case, any error that occurs during the registration of one device has no effect on the registration of any others. When the method completes, *numDevicesDiscovered* again returns the number devices discovered. Registration of those devices proceeds asynchronously. As each device completes registration, it appears in the **Discovered.Uninstalled** subsystem. To limit resource usage during background discovery, the Object Server limits the number of devices that may be registered to 50. You can use the *OnChangeEvent* event to track when the devices have been registered. The event will be fired once for each device that is registered, so you can assume the registration process has not completed until the number of events fired matches the value of the *numDevDiscovered* element. When the *DiscoverDevices* method is halted by an error or its background registration limit, and additional devices remain to be registered, the Object Server's *DiscoveryLimitedFlag* property is set to True. To ensure that all discovered devices have been registered, the *DiscoverDevices* method must be reinvoked. In particular, when registering devices in the background (*backgroundReg* = True), this method must be invoked repetitively until no further devices are discovered. The repeat interval must be sufficiently long to allow the server time to register all devices discovered during the previous *DiscoverDevices* invocation. When the *DiscoverDevices* call is made within an explicit transaction (see *StartTransaction*) and foreground

	registration is used, an error will cause all device registrations to fail. If registration is done in the background, the background registration tasks are performed outside of the scope of the explicit transaction, so a registration failure will only impact the effected device. The other registration tasks will continue to completion (provided no additional errors occur). If the specified *domainId* string is an empty string, the OpenLNS Server will perform the discovery on the zero-length domain. When discovery is performed on a domain other than the application domain, it may not be possible to discover all devices on the network. For example, in a multiple channel system, devices which reside on the far side of a configured router will not be detected because the router will only forward messages addressed on the application domain. In addition, any device using the same addressing information (domain, subnet and Node ID) as the *NetworkServiceDevice* for the *System* will not be discovered. By necessity, discovery messages are addressed on the domain whose devices are to be discovered.
Availability	Local, full, and lightweight clients.
Syntax	*numDevDiscovered* = *systemObject*.**DiscoverDevices**(*backgroundReg, domainId*) **Element** — **Description** *numDevDiscovered* — An Integer value set to the number of devices discovered. *systemObject* — The *System* object to be acted on. *backgroundReg* — A Boolean type specifying whether discovered devices are registered in the background. **TRUE**. Devices are registered in the background as part of the discovery transaction. **FALSE**. Devices are registered in the foreground as part of the discovery transaction. *domainId* — A String specifying the domain on which to discover devices. If the specified *domainId* string is an empty string, the OpenLNS Server will perform the discovery on the zero-length domain.
Added to API	Prior to LNS Release 3.0.

DoEventSync

Summary	Sustains network synchronization.

	This method resynchronizes the network. It should be called every 3 to 5 seconds. This method must be used for event synchronization and missed events to work. If the *SetEventSyncMode* method has been used to turn off event synchronization and missed events are not being used, this method need not be called.
Availability	Local, full, and lightweight clients.
Syntax	*object*.**DoEventSync** **Element** — **Description** *systemObject* — The *System* object which is to be synchronized.
Added to API	Prior to LNS Release 3.0.

DoRestoreOptions

Summary	Restores the current data server options from the persistent database. This method uses the *DsSaveOptions* method to overwrite any values which had been set during the current session. Any options that you modified with this method will not be implemented until you pause and restart the data server. This method will restore all *NetworkVariable* data server options except *DsMonitorTag*, which is not persistent
Availability	Local, full, and lightweight clients.
Syntax	*object*.**DsRestoreOptions** **Element** — **Description** *systemObject* — The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

DsSaveOptions

Summary	Saves the current set of data server options to the persistent database. Data server options are not persistent for *System* objects. For new option values to persist across LNS sessions, you must invoke the *DsSaveOptions* method. This method saves the values which were most recently set, not the values currently in use. This distinction arises because you must pause and restart the data server using the before modified options can be implemented.
Availability	Local, full, and lightweight clients.
Syntax	*object.DsSaveOptions* **Element** — **Description** *object* — The *System* object to be acted on.

Added to API	Prior to LNS Release 3.0.

EndAttachmentEvent

Summary	Disables the *OnAttachmentEvent* for a system for a given object type. To end events for device and router attachments, you must call this method twice: once with the **lcaAppDeviceEvent** parameter, and once with the **lcaRouter** parameter.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.EndAttachmentEvent objectType* **Element** — **Description** *systemObject* — The *System* object to be acted on. *objectType* — An integer determining the object type for which to end attachment events. The possible values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**
Added to API	Prior to LNS Release 3.0.

EndChangeEvent

Summary	Disables the *OnChangeEvent* for a system for a given object type. You must call this method once for every type of object for which change events are to be received.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*.**EndChangeEvent** *objectType* **Element** — **Description** *systemObject* — The *System* object to be acted on. *objectType* — An integer determining the object type for which to end change events. The possible values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**

Added to API	Prior to LNS Release 3.0.

EndCommissionEvent

Summary	Disables the *OnCommission* event for a system for a given object type.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*.**EndCommissionEvent** *objectType* **Element** — **Description** *systemObject* — The *System* object to be acted on. *objectType* — An integer determining the object type for which commission events will be disabled. The possible values for this element, which are contained in the *ConstDeviceEventTypes* constant, are as follows: **0 lcaAppDeviceEvent** **1 lcaRouterEvent**
Added to API	Prior to LNS Release 3.0.

EndLonMarkObjectStatusChangeEvent

Summary	Disables the *OnLonMarkObjectStatusChange* event. You can enable the *OnLonMarkObjectStatusChange* event by invoking the *BeginLonMarkObjectStatusChangeEvent* method. The *OnLonMarkObjectStatusChange* event will then be fired each time an OpenLNS client application changes the status of a *LonMarkObject* on the system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*. **EndLonMarkObjectStatusChangeEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

EndMissedEvent

Summary	Disables the *OnMissedEvent* for a system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*. **EndMissedEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on.

Added to API	Prior to LNS Release 3.0.

EndNodeConnChangeEvent

Summary	Disables the *OnNodeConnChangeEvent* for a system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*. *EndNodeConnChangeEvent* **Element** — **Description** *systemObject* — The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

EndNodeIntfChangeEvent

Summary	Disables the *OnNodeIntfChangeEvent* for a system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*. **EndNodeIntfChangeEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

EndNssIdleEvent

Summary	Disables the *OnSystemNssIdleSystem* a system.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*. **EndNssIdleEvent** *maxIntervalTime* **Element** — **Description** *systemObject* — The *System* object to be acted on. *maxIntervalTime* — A Long value specifying the maximum time between idle events in milliseconds. The valid range for this element is 0–65,535.
Added to API	Prior to LNS Release 3.0.

EndServicePinEvent

Summary	Disables the *OnSystemServicePin* event for a system.
Availability	Local, full, and lightweight clients.
Syntax	*eventTag* = *systemObject*.**EndServicePinEvent** **Element** — **Description** *systemObject* — The *System* object to be acted on. *eventTag* — An Integer representing an allocated event tag.

		This event tag must be the one returned by the *BeginServicePinEvent* method.
Added to API	Prior to LNS Release 3.0.	

EndSession

Summary	Ends a session for methods and properties that must be grouped to avoid intermediate error conditions. All methods invoked and properties written between the *BeginSession* and *EndSession* methods are considered atomic. This method allows your application to create connections more efficiently, and avoid failure scenarios that can occur when devices or routers are moved or changed one-by-one. A given client can have at most one session in progress at a time. A session must be part of an explicit transaction.	
Availability	Local, full, and lightweight clients.	
Syntax	*systemObject.EndSession sessionClass*	
	Element	**Description**
	systemObject	The *System* object to be acted on.
	sessionClass	The session class. Currently, the only valid value is **lcaSessionMove**, which is contained in the *ConstSessionClass* constant.
Added to API	Prior to LNS Release 3.0.	

EndSystemMgmtModeChangeEvent

Summary	Disables the *OnSystemMgmtModeChangeEvent*. The *OnSystemMgmtModeChangeEvent* is fired every time the system's management mode changes. You can enable the *OnSystemMgmtModeChangeEvent* for your application by invoking the *BeginSystemMgmtModeChangeEvent* method. You can use the *MgmtMode* property to read or write to the system management mode.	
Availability	Local, full, and lightweight clients.	
Syntax	*system*.**EndSystemMgmtModeChangeEvent**	
	Element	**Description**
	systemObject	The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.	

ExtensionByHandle

Summary	Retrieves an *Extension* object by its *handle* property. *Extension* records are defined for OpenLNS objects such as the *Channel*, *DeviceTemplate*, *AppDevice*, *Subnet*, *Subsystem*, *LonMarkObject*, and *Router* objects. You can use the *ExtensionByHandle* method of the *System* object to get an *Extension* object in the network database through its *handle* property—regardless of the collection containing the *Extension* object. This may be particularly useful when processing events reported by *OnChangeEvent* with an objectType of *lcaChangeEventExtensions*, and a non-zero *networkHandle*. If the *networkHandle* in the *OnChangeEvent* is 0, use the *ExtensionByHandle* method of the *ObjectServer* instead.
Availability	Local, full, and lightweight clients.
Syntax	*extension* = *system*.**ExtensionByHandle** **Element** — **Description** *Extension* — The *Extension* object. *system* — The *System* object. *handle* — The handle of the *Extension* to be retrieved.
Added to API	OpenLNS.

GetPermission

Summary	Returns the permission value for a specified client. This method is provided to facilitate the testing of permission strings by a local OpenLNS application. Lightweight clients cannot use this method to determine access permission because they will be unable to access the *Network* and *System* objects unless they already have permission to operate on the network.
Availability	Local, full, and lightweight clients.
Syntax	*permValue* = *systemObject*.**GetPermission**(*clientId*) **Element** — **Description** *permValue* — A Long value containing the permission level. A return value of **0** indicates that permission is denied. A return value of **1** indicates that permission is given. *systemObject* — The *System* object to be acted on. *clientId* — A String containing an IP address

	of the local client (for example "192.168.0.3").
Added to API	Prior to LNS Release 3.0.

GetProgramId

Summary	Returns the program ID contained within a specified external interface (XIF) file. You can only use this method with a text-based XIF (.XIF extension). You can not use it with a binary XIF (.XFB extension).
Availability	Local, full, and lightweight clients.
Syntax	*programID* = *systemObject*.**GetProgId**(*XifPath*) **Element** — **Description** *programID* — A String containing the program ID. *systemObject* — The *System* object to be acted on. *XifPath* — A String containing the path to the XIF file.
Added to API	Prior to LNS Release 3.0.

Open

Summary	Opens the system object for use by attaching the system's network interface and setting up the system's OpenLNS Server. This method initializes the *System* object, starts up the underlying OpenLNS Server, and attaches to the LONWORKS network. See the *Open* method of the *ObjectServer* for information on how to initialize the *ObjectServer*. If the application program is not a remote client, the *NetworkInterface* for the System needs to be selected before invoking this method (*System* **.***NetworkServiceDevice***.***NetworkInterface*). When opening a system for the first time, you can set several properties of the *System* object that control the configuration and operation of OpenLNS prior to invoking this method. If you do not set these properties, they will be assigned the following default values by this method: **Property** — **Default** *DomainId* — If *lcaSharedMedia* is specified for the InstallOptions property (below), then a unique 6-byte *DomainId* based on the Neuron ID

	of the network interface is created. The value of the DomainId property cannot be changed if any devices have been commissioned. *InstallOptions* *lcaPrivateMedia* If this method is being invoked by a remote client, the application program needs to manage several additional properties. If the remote connection uses network management authentication, you must set the *AuthenticationKey* property before invoking this method. In addition, this method will set the *Ping* class for this client to Permanent. Anytime after the *Open* method is completed, the application may change the *Ping* class if desired. The presence of repeater and permanent bridge routers in the network path between a remote Full client application host and the Object Server make it impossible for the Object Server to automatically determine the channel on which the remote host resides. In this case, the Full client application must explicitly set a channel by selecting the appropriate channel from the *Channels* property of the *Network* object, and then setting the *RemoteChannel* property to this channel prior to opening the system. This method cannot be invoked as part of an explicit transaction.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.Open* **Element** **Description** *systemObject* The *System* object to be acted on.
Added to API	Prior to LNS Release 3.0.

PrepareToRecoverFromNetwork

Summary	Prepares the system for recovery of the OpenLNS database from the physical network. The *System* object uses two methods to support network recovery: the *PrepareToRecoverFromNetwork* method sets up for recovery and the *RecoverFromNetwork* method performs the actual recovery. The *RecoveryStatus* property allows users to view the state of recovery by returning a *RecoveryStatus* object with recovery information. To recover from the network, follow these steps. Consult the *OpenLNS Programmer's Guide* for more detailed information: 1. Invoke the *Add* method of the *Network* object to create a new *Network*. 2. Set the *NetworkInterface* object contained in the *System*

<table>
<tr><td></td><td>object's NetworkServiceDevice object to identify the network interface for the new network.
3. Call the PrepareToRecoverFromNetwork method. Note that the Network object must be newly created; if objects have been added after creation, an error will result.
4. Call the Open method of the System object. Note that if the recoverNetInterface parameter of the PrepareToRecoverFromNetwork method was set to True, the domain signature will be taken from the network interface device, and not from the System object's DomainId property.
5. Optionally add objects, set properties, and call methods. This includes defining DeviceTemplate objects and setting network timers or other properties of the System object.
6. Call the RecoverFromNetwork method of the System object.
If the recovery operation is aborted (by the computer being turned off, for example), it is possible to resume recovery. To do this, follow these steps:
7. Call the Open method of the Network object on the appropriate network.
8. Call the Open method of the System object.
9. Call the RecoverFromNetwork method of the System object with the resumeRecovery parameter set to True.</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>systemObject.PrepareToRecoverFromNetwork recoverNetInterface
Element — Description
systemObject — The System object to be prepared.
recoverNetInterface — A Boolean type specifying whether OpenLNS will try to recover the network image from the network interface.
TRUE. The domain signature and authentication key are to be recovered from the network interface.
This option is only valid for Layer 5 network interfaces with configurations that are consistent with the network.
If you set this element to True and you are using a</td></tr>
</table>

	Layer 2 network interface, an exception will be thrown. **FALSE**. OpenLNS will read the domain signature from the *DomainId* property. In this case, you should make sure the *DomainId* property is set to a valid value.
Added to API	Prior to LNS Release 3.0.

RecoverFromNetwork

Summary	Recovers the OpenLNS database from the physical network. The *System* object uses two methods to support network recovery: the *PrepareToRecoverFromNetwork* method sets up for recovery and the *RecoverFromNetwork* method performs the actual recovery. The *RecoveryStatus* property allows users to view the state of recovery by returning a *RecoveryStatus* object with recovery information. To recover from the network, follow these steps. Consult the *OpenLNS Programmer's Guide* for more detailed information: 1. Call the *Add* method of the *Network* object to create a new *Network*. 2. Set the *NetworkInterface* object contained in the *System* object's *NetworkServiceDevice* object to identify the network interface for the new network. 3. Call the *PrepareToRecoverFromNetwork* method. Note that the *Network* object must be initialized; if objects have been added after creation, an error will result. 4. Call the *Open* method of the *System* object. Note that if the *recoverNetInterface* parameter of the *PrepareToRecoverFromNetwork* method was set to True, the domain signature will be taken from the network interface device, and not from the *System* object's *DomainId* property. 5. Optionally add objects, set properties, and call methods. This includes defining *DeviceTemplate* objects and setting network timers or other properties of the *System* object. 6. Call the *RecoverFromNetwork* method of the *System* object. If you have modified the attributes of the network image of any of the devices in your network outside of OpenLNS, including the channel ID of any of the devices, the OpenLNS Server may be unable locate that device on the network during recovery. This may cause recovery to fail. 7. After the network recovery has completed, the

<table>
<tr><td></td><td>
MgmtMode property will be set to lcaMgmtModeDeferConfigUpdates (1). You will need to set the property to lcaMgmtModePropagateConfigUpdates (0) before resuming normal operations.

If the recovery operation is aborted (by the computer being turned off, for example), it is possible to resume recovery. To do this, follow these steps:

1. Call the Open method of the Network object on the appropriate network.
2. Call the Open method of the System object.
3. Call the RecoverFromNetwork method of the System object with the resumeRecovery parameter set to True.

Note: The network recovery process should not be considered a replacement for backing up your network database. Although the network recovery process initiated by the RecoverFromNetwork method can recover an entire network, you may discover inconsistencies in the network database depending on the network configuration and the size of the database being recovered.

There are also many parts of an OpenLNS database that are not recoverable, including most object names, connection templates, hub/target relationships, and connections involving Network Service Devices. As a result, you should examine the recovered database before setting the MgmtMode property to lcaMgmtModePropagateConfigUpdates (0) once a recovery has completed.

For more information on this, see the OpenLNS Programmer's Guide.
</td></tr>
<tr><td>Availability</td><td>Local, full, and lightweight clients.</td></tr>
<tr><td>Syntax</td><td>
systemObject.RecoverFromNetwork resumeRecovery, options

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>systemObject</td><td>The System object for the recovered network.</td></tr>
<tr><td>resumeRecover</td><td>A Boolean value specifying whether the process is resuming from a previously started recovery.</td></tr>
<tr><td>options</td><td>A Long indicating the recovery options.

The valid values for this element, which are contained in the ConstRecoveryOptions constant, are as follows:

1</td></tr>
</table>
</td></tr>
</table>

	lcaRecoveryOptSmallNetwork Assume that the network is a small or medium sized network. This causes database recovery to use domain wide broadcast when discovering application nodes. This may not work if the system is very large, as the number of responses from the devices would overwhelm the network. If not set, the recovery process uses subnet broadcast on each of the 255 subnets, each of which must time out before going on to the next. **2 lcaRecoveryOptForceOffline** This option is used to force all devices off-line during the discovery process. This may be useful when recovering networks with high levels of traffic, or when recovering devices that have limited receive transactions and are receiving messages from application devices. **Note***:* Setting this value to **0** will cause neither of the recovery options to be selected.
Added to API	Prior to LNS Release 3.0.

RetryUpdates

Summary	Retries device updates. If a device update failure occurs, you can use this method to retry the updates. A device update failure occurs when OpenLNS is unable to load the information into the physical device when the OpenLNS database has been updated. Many circumstances can cause a device update failure, which is indicated by an exception in the range starting with the **NS, #4030 lcaErrNsWarningFirst** exception, and ending with the **NS, #4089 lcaErrNsWarningLast** exception. For example, an update failure may occur if you make changes to a connection, and one of the devices involved in the connection is not connected to the network or not responding to network management messages. You can set up automatic retries in case of device update failure by setting the *UpdateInterval* property to a non-zero value. If you are receiving persistent update failures for a device, you may need to re-commission the device with the *Commission* method.

	This method cannot be part of an explicit transaction. For more information on explicit transactions, see the *StartTransaction* method. Normally, updates are only attempted while the network management mode (*MgmtMode* property) is set to **lcaMgmtModePropagateConfigUpdates (0).** You can try some updates, however, even while the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**. You can therefore call this method while the network management mode (*MgmtMode* property) is set to either **lcaMgmtModePropagateConfigUpdates (0)** or **lcaMgmtModeDeferConfigUpdates (1)**. See the *PropagateDeviceConfigUpdates*, *CommissionEx*, and *ReplaceEx* methods for the *AppDevice* and *Router* objects more information. **Note**: If you invoke this method while the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1),** only failed updates that occurred while the network management mode was set to **lcaMgmtModeDeferConfigUpdates (1)** will be retried. Failed updates that occurred while the network management mode was set to **lcaMgmtModePropagateConfigUpdates (0)** will not be retried, unless the network management mode is still set to (or has been restored to) the **lcaMgmtModePropagateConfigUpdates (0)** value when you call the *RetryUpdates* method.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject*.**RetryUpdates** **Element** — **Description** *systemObject* — The *System* object for the recovered network.
Added to API	LNS Release 3.0.

SetEventSyncMode

Summary	Sets up resychronization parameters. You should call this method to set up the resynchronization parameters. This includes the setting of automatic resynchronization as well as the setting of the service delay.
Availability	Local, full, and lightweight clients.
Syntax	*systemObject.SetEventSyncMode mode, delay* **Element** — **Description** *systemObject* — The *System* object to be acted on. *mode* — The synchronization mode.

		The possible values for this parameter, which is contained in the *ConstEventSyncModes* constant, are as follows: **0 lcaEventNoSyncMode** Disables automatic event synchronization. If missed events are enabled, an *OnMissedEvent* will be sent for every missed event, including the periodic ping event. Missing a ping event will cause an *OnMissedEvent* to be sent with the *isUnrecoverable* parameter set to **TRUE**, and the *numMissedEvents* parameter set to **0**. **1 lcaEventAutoSyncMode** Enables automatic event synchronization. Recovery will be done automatically within calls to the *DoEventSync* method. An *OnMissedEvent* will only be sent if the Server's "previously sent events" buffer overflows. This buffer can hold up to 20 events. If the remote is out of contact while more than 20 events were generated for it, an *OnMissedEvent* will be generated with the *isUnrecoverable* parameter set to **TRUE** and the *numMissedEvents* parameter equal to some positive integer. This lets the remote device know that it should flush out any caches it maintains via event updates. **2 lcaEventHostSyncMode** This is not currently supported.
	delay	The delay (in milliseconds) prior to delivering a resynchronization request. The default value is **0**. You shoud set the *delay* parameter when there are more than a few remote event subscribers to prevent the network from being flooded with resynchronization

	requests. You only use this parameter if the *mode* parameter is set to *lcaEventAutoSyncMode*.
Added to API	LNS Release 3.0.

StartTransaction

Summary	Starts a transaction for the *System* object. You should call this method in combination with *CommittTransaction* to group method invocations and property modifications as part of the same transaction. If a transaction initiated by your client is already in progress for the *System* when you call this method, an exception will be thrown. You can complete the transaction by calling the *CommittTransaction* method, or you can cancel the transaction by calling the *CancelTransaction* method. When a transaction is canceled, either explicitly or due to a reset, all LONWORKS network and database changes made since the call to *StartTransaction* are reversed. If a transaction is aborted after this method is called, all future services will return an error until the client cancels the transaction using the *CancelTransaction* method. Using transactions can increase performance dramatically in some cases. There is a constant overhead for starting and committing transactions, but no updates are propagated on the network and no changes are written to disk until the transaction is committed. This greatly increases performance when performing multiple coordinated changes to your database. Transactions, however, consume memory, and if an excessive number of changes are made during a single transaction, the resulting overhead may slow down the overall process. You should use transactions to synchronize changes so that only one client can have a transaction open at a time. While one client is making a change in a transaction, other clients will be paused while waiting to gain the transaction lock. Some database operations will implicitly create a transaction, and they will therefore wait for the other client to complete its transaction before that operation will continue. Transactions can also be used to perform "atomic" sets of changes, which are changes that require a series of calls to LNS to be made. For example, moving an application device or router is an operation that could be performed within a transaction. If any of the calls made during such a transaction fails, all of the changes made up to that point can be undone by simply cancelling the transaction. For more detailed information on when you should use

	transactions, see the *Using Transactions and Sessions* section in the *OpenLNS Programmer's Guide*. You can use the *OnSystemNssIdle* event to allow your application to execute code while lengthy transactions are being performed. For example, you can use the *ServiceStatus* property to find out if this client is waiting on a transaction started by another client, or you can call *CancelTransaction* to cancel the operation. For more information on this, see the *OnSystemNssIdle* event.
Availability	Local, full, lightweight, and independent clients.
Syntax	*systemObject*. **StartTransaction** **Element** — **Description** *systemObject* — The *System* object whose transaction will be canceled.
Added to API	Prior to LNS Release 3.0.

WinkByNeuronId

Summary	Enables a device to be winked by Neuron ID, which lets you physically identify the device.
Availability	Local, full, and independent clients.
Syntax	*systemObject*.**WinkByNeuronId** *NeuronId* **Element** — **Description** *systemObject* — The *System* object containing the *AppDevice* to be winked. *NeuronId* — A string containing the Neuron ID of the device to be winked.
Added to API	LNS Release 3.0.

Properties

The *System* object contains the following properties:

- *Accounts*
- *ActivationLicense*
- *ApplicationHandle*
- *ApplicationName*
- *AuthenticationKey*
- *ClassId*
- *ClientId*
- *CommissionedDeviceCount*
- *ComponentApps*
- *Connections*
- *CurrentAccount*
- *CurrentDeviceCount*
- *CustomerId*
- *DebugTraceFlag*

- *Description*
- *DiscoveryInterval*
- *DiscoveryLimitedFlag*
- *DomainId*
- *DsPollInterval*
- *DsPriority*
- *DsRepeatTimer*
- *DsRetries*
- *DsRetryCount*
- *DsTxTimer*
- *Extensions*
- *FileTransfer*
- *Handle*
- *HostTimer*
- *ImportDirectory*
- *InstallOptions*
- *IsOpen*
- *LastError*
- *LaunchLcaServerFlag*
- *LdrfCatalogPath*
- *LdrfLanguages*
- *MgmtMode*
- *Name*
- *NetworkResources*
- *NetworkServiceDevice*
- *NssDbVersion*
- *Parent*
- *PermissionString*
- *PingIntervals*
- *RecoveryStatus*
- *RegisterServicePin*
- *RemoteChannel*
- *RepeatTimer*
- *ResourceLanguageId*
- *RetryCount*
- *SecurityLevel*
- *ServiceStatus*
- *State*
- *Subnets*
- *Subsystems*
- *TemplateLibrary*
- *TxTimer*
- *UncommissionedDeviceCount*
- *UninstalledDeviceCount*
- *UpdateInterval*

Accounts

Summary	Contains the *Accounts* object associated with the *System* object. This property and the associated *Account* and *Accounts* objects are reserved for future use.

Availability	Local, full, and lightweight clients.
Syntax	*acctsCollection* = *sysObject*.**Accounts** **Element** **Description** *acctsCollection* The *Accounts* collection to be returned. *sysObject* The *System* object to be acted on.
Data Type	*Accounts* collection object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ActivationLicense

Summary	Returns an *ActivationLicense* object that contains a snapshot of the activation license on the OpenLNS Server at the time it was first accessed. The properties of the *ActivationLicense* object will not change if you re-read them, even if the license becomes invalid in the meantime. To get an updated snapshot of the license information, call the *Refresh* method or release this object and acquire a new one by re-reading this property. You can only read this after you open the system; however, you can access it even if the system is in "licensing failure mode". This object will be returned regardless if the license is valid for this computer or not.
Availability	Local, full, and lightweight clients.
Syntax	*activationLicense* = *systemObject*.**ActivationLicense** **Element** **Description** *activationLicense* An *ActivationLicense* object representing a snapshot of the activation license on the OpenLNS Server. *systemObject* The *System* object to be acted on.
Data Type	*ActivationLicense* object.
Read/Write	Read only.
Added to API	OpenLNS.

ApplicationHandle

Summary	The unique identifier assigned to this instance of the Object Server by the NSS Engine. The *ApplicationHandle* property is used internally by the Object Server to register service requests and event handles with the NSS.

Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *sysObject*.**ApplicationHandle** **Element** **Description** *returnValue* The handle of the application. *sysObject* The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ApplicationName

Summary	The application name used by the Object Server when registering for an application handle with the NSS. If a name is not provided by the application using the Object Server, a default name is used. The default is *LCAx* where *x* = 1, 2, 3, and so on, and it is assigned by the NSS. If the application sets this property, it must be set before you call the *System*.*Open* method.
Availability	Local, full, and lightweight clients.
Syntax	*nameValue* = *sysObject*.**ApplicationName** **Element** **Description** *nameValue* The name of the application. *sysObject* The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

AuthenticationKey

Summary	Identifies the authentication key associated with the system, if any. All devices and router sides that have the *AuthenticationEnabled* property set to True will have this key installed by the Object Server or should have it installed by some external means (see the *System* object's *SecurityLevel* property). When this property is changed, all devices with *AuthenticationEnabled* set to True are incrementally updated to contain the new key. Authentication keys may be stored as hexadecimal values of 12 characters for 48-bit authentication (for example "a327ff27ba24"), or 24 characters for 96-bit authentication (Open Media Authentication [OMA]); for example "a327ff27ba24 a327ff27ba24". Each character represents one

	hexadecimal digit of the key. A key of ffff ffff ffff or ffff ffff ffff ffff ffff ffff indicates that authentication is disabled. Remote Full and Lightweight client applications that are connecting to an authenticated network must set this property before opening the system. If the application uses the wrong authentication key, the client must close the *System*, *Network*, and *ObjectServer* objects, re-acquire and re-open the *ObjectServer* and *Network* objects, and the re-acquire the *System* object. The application then must set the *AuthenticationKey* property to the correct authentication key, and re-open the system with the *Open* method. For more information on opening systems and setting system parameters, see the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*keyValue* = *systemObject*.**AuthenticationKey** **Element** **Description** *keyValue* The authentication key. *systemObject* The *System* object.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *System* object in the *ConstClassIds* constant: **3 lcaClassIdSystem** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.

ClientId

Summary	Contains the *ClientId* associated with the *System* object by the Object Server engine. If the OpenLNS application needs to directly invoke the services of the Object Server engine, the ClientId property may be needed depending upon the nature of the service invoked.
Availability	Local, full, and lightweight clients.
Syntax	*clientIdValue* = *systemObject*.**ClientId** **Element** — **Description** *clientIdValue* — The client Id assigned to the *System* object as a Long. *systemObject* — The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

CommissionedDeviceCount

Summary	Returns the number of commissioned devices restored by network recovery. At the completion of network recovery, devices are grouped into three categories. • Installed devices that were recovered are counted by the *CommissionedDeviceCount* property. • Installed devices that could not be recovered, and had to be uninstalled, are counted by the *UninstalledDeviceCount* property. • Devices that were discovered by network recovery but had not previously been installed are counted by the *UncommissionedDeviceCount* property. If the OpenLNS application needs to directly invoke the services of the Object Server engine, you may need the *ClientId* property depending upon the service you invoked.
Availability	Local, full, and lightweight clients.
Syntax	*numDevices* = *systemObject.CommissionedDeviceCount* **Element** — **Description** *numDevices* — The number of restored devices. *systemObject* — The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

ComponentApps

Summary	Contains the *ComponentApps* collection object associated with the specified *System* object. The *ComponentApps* collection is a list of LNS plug-in commands that are associated with a particular object type.
Availability	Local, full, and lightweight clients.
Syntax	*appsCollection* = *object.ComponentApps* **Element** — **Description** *appsCollection* — The *ComponentApps* collection to be returned. *object* — The *System* object to be acted on.
Data Type	*ComponentApps* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Connections

Summary	Returns the *Connections* object associated with the specified *System* object. Use this property to get the *Connections* object associated with the *System*, which in turn contains the *NetworkVariables* and *MessageTags* collection objects that represent all of the connections in the system.
Availability	Local, full, and lightweight clients.
Syntax	*connectionsObject* = *systemObject.Connections* **Element** — **Description** *connectionsObject* — The returned *Connections* object. *systemObject* — The *System* object to be acted on.
Data Type	*Connections* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

CurrentAccount

Summary	Contains the *Account* object representing the active licensing account. This property and the associated *Account* and *Accounts* objects are reserved for future use.
Availability	Local, full, and lightweight clients.

Syntax	*acctObject* = *systemObject*.**CurrentAccount** **Element** **Description** *acctObject* The current *Account* object. *systemObject* The *System* object to be acted on.
Data Type	*Account* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

CurrentDeviceCount

Summary	Returns the number of *AppDevice* objects currently defined, including network service devices.
Availability	Local, full, and lightweight clients.
Syntax	*deviceCountValue* = *systemObject*.**CurrentDeviceCount** **Element** **Description** *deviceCountValue* The number of devices currently defined. *systemObject* The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

CustomerId

Summary	Returns the current *CustomerId*, as set in the *SetCustomerInfo* method of the *ObjectServer*.
Availability	Local, full, and lightweight clients.
Syntax	*custIdValue* = *systemObject*.**CustomerId** **Element** **Description** *custIdValue* The customer ID to be returned *systemObject* The *System* object to be acted on.
Data Type	String.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DebugTraceFlag

Summary	Enables the trace debugging tool.
Availability	Local, full, and lightweight clients.
Syntax	*debugFlag* = *systemObject*.**DebugTraceFlag**

	Element **Description** *systemObject* The *System* object to be acted on. *debugFlag* A Boolean value that indicates whether trace should be turned on. **TRUE**. The trace should be turned on. Trace messages will be stored in the file "lcatrace.txt". **FALSE**. Trace should not be turned on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *System* object.
Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** **Description** *stringValue* A string description of the *System* object. *object* The *System* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

DiscoveryInterval

Summary	Indicates the rate at which the Object Server scans the network for newly attached, unconfigured devices. A device must be unconfigured to be discovered by the background discovery mechanism. Setting the interval value to **0** disables the automatic discovery of unconfigured devices. The default value for this property is based on the value of the *InstallOptions* property when the system is first opened: • Setting the *InstallOptions* property to **lcaSharedMedia (2)** disables background discovery by setting this property to **0**. • Setting the *InstallOptions* property to **lcaPrivateMedia**

	(4) sets this property to **180** seconds.
Availability	Local, full, and lightweight clients.
Syntax	*intervalValue* = *systemObject*.**DiscoveryInterval** *systemObject*.**DiscoveryInterval** = *intervalValue**Element** — **Description** *intervalValue* — The new device discovery interval (in seconds). An interval of **0** turns off background discovery process. Valid values are 1 to 65,534. *systemObject* — The *System* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

DiscoveryLimitedFlag

Summary	Indicates that device discovery was halted due to resource limitations. The *DiscoverDevices* method must be reinvoked. You can perform device discovery in the foreground by invoking the *DiscoverDevices* method. If you invoke this method with the *backgroundReg* parameter set to True, discovered devices will be queued for registration in background transactions. The OpenLNS Server limits the number of background registation tasks to 50 to prevent a significant resource drain. This property is unaffected by background discovery.
Availability	Local, full, and lightweight clients.
Syntax	*discoveryStopped* = *systemObject*.**DiscoveryLimitedFlag****Element** — **Description** *discoveryStopped* — A Boolean value that indicates whether device discovery was halted due to the registration limit. The *DiscoveryLimitedFlag* is application exclusive. This means that every OpenLNS application has its own independent flag **TRUE**. Device discovery was halted due to the registration limit. You must re-invoke the *DiscoverDevices* method to ensure that no undiscovered devices remain on the network.

	FALSE. Device discovery has completed execution without being halted or the application has exited. *systemObject* The *System* object to be acted on
Data Type	Boolean.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

DomainId

Summary	Identifies the domain ID. The domain ID is stored as a string of hexadecimal digits. For example, a 3-byte domain ID would be represented like this: "32a0cf". Domain IDs can be 1, 3, or 6 bytes in length. When setting the *DomainId* property, any characters beyond those required are ignored. Changes to the domain ID will automatically be propagated to all application devices and routers unless one or more devices is using authentication. In that case, an exception will be returned. The ability to set this property may be useful in the predefined components installation scenario involving many similarly configured sites. In such a scenario, it is more efficient to load an OpenLNS database containing basic configuration into every site, and change the domain ID of the site immediately prior to commissioning, than to create each database separately. The following is an example sequence: 1. Create a basic configuration database by initializing the Object Server, setting the *System* object's *MgmtMode* property to **lcaMgmtModeDeferConfigUpdates (1)**, creating program templates by importing binary external interface files, defining devices (without Neuron Ids), and adding connections. Save the database by copying the network database directory. 2. During commissioning, load the database by coping the database above and adding it to the global database. 3. Set the **DomainId** property to the domain ID for the site. This must be a unique value. One way to ensure that the *DomainId* property is assigned a unique value is to set the *InstallOptions* property to **lcaSharedMedia** before creating the network. This will result in the OpenLNS Object Server assigning a value equivalent to the Neuron ID of the OpenLNS Server computer to the *DomainId* property, which is guaranteed to be a unique value. 4. Set the *System* object's *MgmtMode* property to **lcaMgmtModePropagateConfigUpdates (0)**.

	5. Use service pin, find, or other manual means of obtaining the actual Neuron ID of each device. 6. Iterate through the **Discovered.Uninstalled Subsystem** object to get the list of devices in the system that require their network images to be updated. If the device's Neuron ID has not been specified, acquire it using the service pin, find/wink, or manual entry methods. *Commission* the device using the *Commission* method. 7. Add and connect any additional devices that are not defined by the basic configuration. Attempting to read the *DomainId* of a newly created system will result in an exception. Once the system has been closed and re-opened, the *DomainId* can be read normally..
Availability	Local, full, and lightweight clients.
Syntax	*domainIdValue* = *systemObject*.**DomainId** **Element** — **Description** *domainIdValue* — The system's domain ID. *systemObject* — The *System* object.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

DsPollInterval

Summary	Specifies the poll or throttle interval (in tenths of a second) used by the Data Server. For remote clients using the shared Data Server mode, this property specifies the batch update rate. The *DsPollInterval* property specifies either a poll or throttle interval, depending on the context in which it is used. • For polled network variables, the property specifies the interval between polls. Setting the value to **0** disables polling. • For bound network variables, the property specifies the minimum interval which must elapse before the Data Server will generate an update event. The throttling interval may be used to regulate the rate in which *OnNetworkVariableUpdate* events occur. Remote client applications that use shared access mode receive batched data updates. Instead of receiving individual network variable value updates over the IP network, the client receives a regular update containing the updated values for the monitored network variables. In this scenario, the *System* object's *DsPollInterval* property specifies the batch update rate for each network variable. If you are

	developing a remote application that uses shared access mode (meaning that the DsMode property is set to **lcaDsModeShared**), you cannot set the *NetworkVariable* object's *DsPollInterval* property. An exception will be raised if you attempt to do so. The *System* object's *DsPollInterval* property sets the default value, which is applied when a *NetworkVariable* object's *DsPollInterval* property is left unspecified. Once an application has explicitly set the value of the *DsPollInterval* property of the *System* object, it will not be reverted back to the default value by OpenLNS, and must be maintained manually from that point on.
Availability	Local, full, and lightweight clients.
Syntax	*intervalValue* = *object*.**DsPollInterval** **Element** — **Description** *intervalValue* — Poll or throttle interval, in tenths of a second. The allowed range is 0 to 33554431 (0x01FFFFFF). This maximum value corresponds to a time of approximately 38.8 days. The default value is **10** (one second) for the *DsPollInterval* property. The system's current *DsPollInterval* value establishes the default that will be used by newly created *NetworkVariable* objects whose *DsPollInterval* value is left unspecified. *object* — The *System* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

DsPriority

Summary	Specifies whether the Data Server polls or updates a network variable using priority messages. **Note**: This property is deprecated because it was only useful with single-point monitoring. You should use temporary monitor sets instead of single-point monitoring. When doing so, you can use the *Priority* property to determine the priority assigned to each message. For more information on temporary monitor sets, see the *OpenLNS Programmer's Guide*.
Availability	Local, full, and lightweight clients.
Syntax	*priorityFlag* = *object*.**DsPriority** **Element** — **Description**

	priorityFlag	The priority flag value. **TRUE**. The Data Server uses priority messaging to update a network variable. **FALSE**. The Data Server polls a network variable. This is the default.
	object	The *System* object to be acted on.
Data Type	Boolean.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

DsRepeatTimer

Summary	Sets the interval between repeated transmissions for messages sent using the unacknowledged/repeat message service. This only applies to messages sent during monitor and control operations for temporary monitor sets. To set this value for network management messages and permanent monitor and control operations, use the *RepeatTimer* property. The default value for this property is determined based on the network topology. You should not modify this default value. If the default value is not suitable for your application, it is recommended that you use the *Delay* property of each *Channel* object on the network to ensure that each message is sent at the correct interval.
Availability	Local, full, and lightweight clients.
Syntax	*repeatTimerValue* = *systemObject*.**DsRepeatTimer**

Element	**Description**
retryCountValue	The encoded value of the repeat timer. This property accepts a range of encoded values of 0–15. The encoded values are for this property are as follows:

Encoded Value	**Seconds**
0	0.016
1	0.024
2	0.032
3	0.048
4	0.064
5	0.096
6	0.128
7	0.192

		8	0.256
		9	0.384
		10	0.512
		11	0.768
		12	1.024
		13	1.536
		14	2.048
		15	3.072
		You can write the value **254** to the property at any time to restore it to the default.	
		To set the number of repeats that will be sent, write to the *RetryCount* property of the *System* object.	
	systemObject	The *System* object to be acted upon.	
Data Type	Integer.		
Read/Write	Read/write.		
Added to API	LNS Release 3.20.		

DsRetries

Summary	Specifies the retry count to use for messages sent using the acknowledged, request/response, or repeated message services.	
Availability	Local, full, and lightweight clients.	
Syntax	*retryValue* = *object*.**DsRetries**	
	Element	**Description**
	retryValue	The number of retries. The default value is **4**.
		This parameter sets the default retry count that is applied to message monitor points in temporary monitor sets.
		Invoke the *DsSaveOptions* method to save the property value into the persistent database. You can use the *RetryCount* property to set the retry count for network management messages.
	object	The *System* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

DsRetryCount

Summary	Sets the maximum number of times to retry messages sent using the request/response, unacknowledged/repeat, or acknowledged message services. This applies to messages sent during monitor and control operations for temporary monitor sets only.
Availability	Local, full, and lightweight clients.
Syntax	*retryCountValue* **=** *systemObject*.**DsRetryCount** **Element** — **Description** *retryCountValue* — The retry count value. The value of this property sets the retry count used by the Object Server (and connections that use the default System value) for temporary monitor set operations that use the service types listed above. This value must be between 0 and 15. • You can set the interval for retry messages sent using the acknowledged or request messages for temporary monitor set operations by writing to the *DsTxTimer* property. • You can set the interval for repeat messages sent using the unacknowledged/repeat message service for temporary monitor set operations by writing to the *DsRepeatTimer* property. **Note**: This property applies to monitor and control messages for temporary monitor set operations only. • To set this value for network management messages, use the *SystemRetryCount* property of the object. To set this value for monitor set operations involving permanent monitor sets, use the *RetryCount* property of the *ConnectDescTemplate* object that is used by connections involving your permanent monitor sets. *systemObject* — The *System* object to be acted upon.
Data Type	Integer.

Read/Write	Read/write.
Added to API	LNS Release 3.20.

DsTxTimer

Summary	Sets the interval between retries for acknowledged (**lcaSvcAckd)** and request (**lcaSvcRequest**) messages sent during monitor set operations involving temporary monitor sets. To set this value for network management messages, use the *TxTimer* property. To set this value for monitor set operations involving permanent monitor sets, use the *TransmitTimer* property of the *ConnectDescTemplate* object used by connections involving your permanent monitor sets. The default value for the *DsTxTimer* property is determined based on the network topology. You should not modify this default value. If the default value is not suitable for your application, you should use the *Delay* property of each *Channel* object on the network to ensure that each message is sent at the correct interval.
Availability	Local, full, and lightweight clients.
Syntax	*timerValue* = *systemObject.DsTxTimer* **Element** — **Description** *timerValue* — The encoded value of the transmit timer. This property accepts a range of encoded values of 0–15. The encoded values are for this property are as follows: **Encoded** *Value* — **Seconds** 0 — 0.016 1 — 0.024 2 — 0.032 3 — 0.048 4 — 0.064 5 — 0.096 6 — 0.128 7 — 0.192 8 — 0.256 9 — 0.384 10 — 0.512 11 — 0.768 12 — 1.024 13 — 1.536 14 — 2.048

	15 3.072 You can write the value **254** to the property at any time to restore it to the default. To set the number of retries that will be sent for these message types, write to the *DsRetryCount* property. *systemObject* The *System* object to be acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.20.

Extensions

Summary	Contains the *Extensions* collection object associated with the specified *System* object. This property returns an *Extensions* collection. The objects in this collection represent user data reserved for manufacturers. Each object is identified with a unique identifier set by the manufacturer
Availability	Local, full, lightweight, and independent clients.
Syntax	*extensionsColl* = *object.Extensions* **Element** **Description** *extensionsColl* The *Extensions* collection object. *object* The object whose *Extensions* collection is being returned.
Data Type	*Extensions* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

FileTransfer

Summary	Returns a new *FileTransfer* object for use in establishing a file transfer session. Every call to get the *FileTransfer* object returns a fresh *FileTransfer* object whose properties are set equal to the defaults. As a result, OpenLNS must retain references to the *FileTransfer* objects with which it intends to perform multiple operations. An application may still create multiple references to a given *FileTransfer* object. Additionally, an application may maintain multiple unique *FileTransfer* objects at any given time.
Availability	Local, full, and lightweight clients.

Syntax	*fileTransObject* = *systemObject*.*FileTransfer* **Element** **Description** *fileTransObject* The *FileTransfer* object. *systemObject* The *System* object to be acted on.
Data Type	*FileTransfer* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Handle

Summary	Contains the handle associated with the *System* object. An OpenLNS Object that is part of a collection is assigned an index corresponding to its position within that collection. This index may be used when invoking the *Item* property. Some OpenLNS Objects are tracked internally by the OpenLNS Server using a unique handle.
Availability	Local, full, and lightweight clients.
Syntax	*returnValue* = *object*.**Handle** **Element** **Description** *returnValue* The NSS handle of the *System* object. *object* The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

HostTimer

Summary	The maximum number of seconds the system will wait for a response message from the host application after a management request has been sent.
Availability	Local, full, and lightweight clients.

Syntax	*hostTimerValue* = *systemObject*.**HostTimer** **Element** — **Description** *hostTimerValue* — The host timer value in seconds. This property applies to network management messages only. This property accepts a range of values from 0 to 65,534 seconds. The default value is **15** seconds. You can write the value **65535** to this property without an exception being thrown. The value of the property, however, will not be changed from its current value. *systemObject* — The *System* object to be acted upon.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

ImportDirectory

Summary	The default directory in which the *Import* method expects to find external interface (XIF) files (**.XIF** and **.XFB** extensions).
Availability	Local, full, and lightweight clients.
Syntax	*pathName* = *systemObject*.**ImportDirectory** **Element** — **Description** *pathName* — The path name of the Import directory as a string. If you only specify a filename without a pathname as a parameter to the *Import* method, that file is expected to be found in the directory specified by this property. *systemObject* — The *System* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

InstallOptions

Summary	Specifies the system install options, and indicates whether the system uses private or shared media. Disable automatic discovery and pinging by setting the

	Disable and *PingIntervals* properties to 0. Disable automatic service pin registration by setting the system's *RegisterServicePin* property to False.
Availability	Local, full, and lightweight clients.
Syntax	*pathName* = *systemObject*.**ImportDirectory** **Element** — **Description** *optionValue* — The value of the selected option. The valid values for this property, which are contained in the *ConstInstallOptions* constant, are as follows: **2 lcaSharedMedia** Indicates that the system may potentially share the media with other independently managed systems. This is typically the case with power line or RF. If the value is set to **lcaSharedMedia**, the *DomainIdSystem* property will automatically be set to the same value as the *NeuronId* property of the *NetworkServiceDevice*. This ensures that no two networks operating in shared media will have the same domain. It also sets the *discovery interval* to **0**, sets all the *ping intervals* to **0**, and sets the *RegisterServicePin* property to False. **4 lcaPrivateMedia** This is the default value. Indicates that no other systems share the media. This is typically the case with a twisted pair network in which you do not intend by design to allow other independently managed systems onto the same physical media. When this option is specified, LNS enables background discovery by initializing the *DiscoveryInterval* property to **180** seconds. Similarly, background node pinging is enabled for mobile nodes by setting the interval to **60** seconds and service pin registration is enabled. Setting this property after the system has been initially opened has no effect;

		therefore, your application should set this property before calling the *Open* method of the *System* object for the first time. If you open a system with this property set to **lcaPrivateMedia (4)** and later decide you want the system to use shared media, you can accomplish this by setting the *DomainIdSystem* property of the *System* object to match the *NetworkServiceDevice.NeuronId* property of the OpenLNS Server computer.
	systemObject	The *System* object for which the options apply.
Data Type	Integer.	
Read/Write	Read/write for Local clients. Read-only for Remote and Lightweight clients.	
Added to API	Prior to LNS Release 3.0.	

IsOpen

Summary	Indicates whether the specified *System* object is currently open.	
Availability	Local, full, lightweight, and independent clients.	
Syntax	*isOpenFlag = Object*.**IsOpe**	
	Element	**Description**
	isOpenFlag	Boolean value.
	Object	A Boolean value indicating whether the *System* object is currently open. **TRUE**. The *System* object is currently open. **FALSE**. The *System* object is currently closed.
Data Type	Boolean.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

LastError

Summary	Returns the last error generated in an *Error* object. The *LastError* property is transient, meaning that the error data is not stored in the OpenLNS database. When a particular application is closed, the *LastError* information is lost. Error information does not persist across program

	invocations.
Availability	Local, full, and lightweight clients.
Syntax	*lastErrorObject* = *systemObject*.**LastError** **Element** — **Description** *errorObject* — The last error generated by the system *systemObject* — The *System* object to be acted upon.
Data Type	*Error* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

LaunchLcaServerFlag

Summary	Indicates whether the OpenLNS Server should be launched when the *Open* method of the *System* object is called.
Availability	Local, full, and lightweight clients.
Syntax	*debugFlag* = *systemObject*.**LaunchLcaServerFlag** **Element** — **Description** *systemObject* — The *System* object to be acted on. *debugFlag* — A Boolean value that indicates whether the OpenLNS Server should be launched when the *System*.*Open* method is called. **TRUE**. The OpenLNS Server should be launched when the system is opened. The OpenLNS Server will automatically be shut down when the last client using the server is closed. **FALSE**. The OpenLNS Server should not be launched when the system is opened The OpenLNS Server process is required to allow OpenLNS Remote Clients of all types to connect. If you do not want remote clients to connect, set this property to False (before invoking the *System* object's *Open* method) to conserve computer resources. The OpenLNS Server can also be manually launched by starting the OpenLNS Server

	Application available from the Echelon OpenLNS Utilities program folder.
Data Type	Boolean.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

LdrfCatalogPath

Summary	Obtains the full path for the location of the LonMark device resource file catalog (**ldrf.cat**). You should not modify this path because the resource file catalog should always be stored in the **LonWorks/Types** folder. The *LdrfCatalogPath* property points this path by default; therefore, you should not write to this property under any circumstances. See the *LonMark Resource File Catalog Help* and the *Device Resource File Developers Guide* Setting this property in the *ObjectServer* object overrides the value in the *System* object. This must be done by a Local client application before you call the *ObjectServer* object's *Open* method. However, you should not write to this property. If you do write to the *LdrfCatalogPath* property, leave the standard resource files in the **LonWorks/Types** folder, and then create a copy of the standard resource files in the new folder referenced by the *LdrfCatalogPath* property
Availability	Local, full, lightweight, and independent clients.
Syntax	*pathName = object.LdrfCatalogPath* **Element** — **Description** *object* — The *System* object to be acted on. *pathName* — The location of the LDRF catalog, which contains the locations of the standard and user-defined resource files, and the associated files that are required for data formatting. The *pathName* cannot include the semi-colon character (;).
Data Type	String.
Read/Write	Read and write for Local client applications. Read only for Full and Lightweight client applications.
Added to API	LNS Release 3.0.

LdrfLanguages

Summary	Contains the *LdrfLanguages* collection representing the languages known by OpenLNS. You can only access this property after you have *opened* the Object Server.
Availability	Local, full, lightweight, and independent clients.
Syntax	*ldrfLang = osObject.LdrfLanguages* **Element** **Description** *ldrfLang* The *LdrfLanguages* collection. *osObject* The *System* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

MgmtMode

Summary	Determine's the OpenLNS Object Server's network management mode. When a remote full client connects to a network, the configuration of its *NetworkServiceDevice* must be updated in order for the connection to succeed. If the *NetworkServiceDevice* had been previously added to the system, and no subsequent changes to its configuration were that need to be coordinated with the other devices on the network, the configuration of the *NetworkServiceDevice* will be updated and the client will open the network. This will happen regardless of the setting of the *MgmtMode* property. However, if the *NetworkServiceDevice* had not been previously added to the system, or its configuration has changed in a way that is not consistent with the other physical devices on the network, then the network management mode will be briefly set to **lcaMgmtModePropagateConfigUpdates (0)** while the connection takes place. In order for this to happen, the *AllowPropagateModeDuringRemoteOpen* property must be set to True. Afterwards, the network management mode will be restored to **lcaMgmtModeDeferConfigUpdates (1)**. If the *AllowPropagateModeDuringRemoteOpen* property is set to False in this case, the connection will fail, and the **NS, #31 lcaErrNsDeferConfigUpdatesMgmntMode** exception will be generated. You can use the *OnSystemMgmtModeChangeEvent* event to track changes to the value of this property.
Availability	Local, full, and lightweight clients.
Syntax	*modeType* = *systemObject.MgmtMode* **Element** **Description**

	modeType	The Object Server management mode. The enumerated values for this parameter, which are contained in the *ConstMgmtModes* constant, are as follows: **0 lcaMgmtModePropagateConfigUpdates** Network configuration changes are applied to both the OpenLNS database and the physical devices. Setting the management mode to this value also starts the background device discovery processes. This causes the Object Server to update the network image of any device which was modified while the network management mode was set to **lcaMgmtModeDeferConfigUpdates (1)**. **1 lcaMgmtModeDeferConfigUpdates** All network configuration changes are applied only to the OpenLNS database, not to the physical devices. In addition, discovery and background updates are turned off. In this mode, device configuration changes are queued for later processing. **Notes**: You can use the *PropagateDeviceConfigUpdates* method to apply device-only configuration changes to a physical device while the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**. For more information, see the *PropagateDeviceConfigUpdates* method. The *ReplaceEx* and *CommissionEx* methods provide options to propagate changes to a physical device when commissioning or replacing a device while the network management mode is set to **lcaMgmtModeDeferConfigUpdates (1)**. For more information, see the *ReplaceEx* and methods.
	systemObject	The *System* object to be acted on
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

Name

Summary	Specifies the name of an object as a character string. This property is case sensitive. Searches by name must match case. This property can be a maximum of 85 characters long, but it may not contain the forward slash (/), back slash (\), period (.), and colon (:) characters.
Availability	Local, full, lightweight, and independent clients. Note that some objects that contain this property are not available to Independent clients.
Syntax	*stringValue* = *object*.**Name** **Element** — **Description** *stringValue* — The name of the object. *object* — The object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API.

NetworkResources

Summary	Provides access to important network resource information for the system, including the number of exclusive and sharable selectors available on the sytem, the number of subnets and group IDs allocated on the system, and the number of *AppDevices* and *Routers* that have been installed on the system. The *System* must be open when you read this property. If it is not open, the **LCA#67 lcaErrSystemNotOpen** exception will be thrown. This property returns a *NetworkResources* object, which contains a group of read-only properties that contain information you may find useful when determining the network resources that are available on your LONWORKS system. For more information, see the *NetworkResources* object.
Availability	Local, full, and lightweight clients.
Syntax	*modeType* = *systemObject.MgmtMode* **Element** — **Description** *resourcesObject* — The *NetworkResources* object returned by the property. *systemObject* — The *System* object to be acted on

Data Type	*NetworkResources* object.
Read/Write	Read only.
Added to API	LNS Release 3.20.

NetworkServiceDevice

Summary	Returns the network service device, either an OpenLNS Server or NSI, attaching the OpenLNS ActiveX control to the network. The *NetworkServiceDevice.NetworkInterface* property of this object must be set before network communications are possible.
Availability	Local, full, and lightweight clients.
Syntax	*nsdObject* = *systemObject.NetworkServiceDevice* **Element** — **Description** *systemObject* — The *System* object to be acted on. *nsdObject* — The *NetworkServiceDevice* object.
Data Type	*NetworkServiceDevice* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

NssDbVersion

Summary	Indicates the version number of the NSS database.
Availability	Local, full, and lightweight clients.
Syntax	*dbVersion* = *systemObject*.**NssDbVersion** **Element** — **Description** *dbVersion* — The version number of the NSS database. The value of this parameter is incremented when the database is modified. A database modification followed by a backout will not result in the version number being restored, rather it will be bumped twice. In other words, a backout constitutes a new version rather than a return to a previous version. Version numbers are 16-bits and thus may wrap around. *systemObject* — The *System* object to be acted on.
Data Type	Integer.
Read/Write	Read only.

Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

PermissionString

Summary	Contains the permission string which controls system access for Lightweight clients. The permission string specifies the access permission for Lightweight clients. It consists of a concatenated list of permission specifications (spec) separated by commas or line feeds. Each permission spec defines a rule which grants or denies access to one or more IP addresses. An individual spec has the form `<IP Address> <IP Mask>, <Permission>`, where permission has the value 0 (deny) or 1 (allow). For example: `192.168.0.5 255.255.255.255, 1` A permission spec applies to all Lightweight clients with an IP address that satisfies the rule `(<Remote Client IP Address> & <IP Mask>) == <IP Address>`. In other words, the client's IP address is logically AND'ed with the IP mask, and the result is compared to the IP Address in the permission spec. If they match, the specified permission value is applied to the client. Thus, the example above only provides permission to a client on a computer with the IP address 192.168.0.5. The Object Server evaluates permission specs in the order that they appear within the permission string, returning the

	permission value given by the first matching spec. This feature may be used to implement sophisticated permission scenarios. Some examples follow below.
	This spec gives permission to all IP address starting with 192.168:
	`192.168.0.0  255.255.0.0,  1`
	This spec denies access to IP 192.168.0.3:
	`192.168.0.3  255.255.255.255,  0`
	The following spec has no IP address which satisfies its rule, so an error will be generated upon assignment:
	`192.168.1.2  255.255.0.0,  1`
	The following two specs provide access to all IP address beginning with 192.168, except those starting with 192.168.5:
	`192.168.5.0  255.255.255.0,  0`
	`192.168.0.0  255.255.0.0,  1`

Availability	Local clients.
Syntax	*permString* = *systemObject*.**PermissionString** **Element** — **Description** *permString* — The returned permission string. *systemObject* — The *System* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

PingIntervals

Summary	Allows the ping intervals for the system to be changed using the *PingIntervals* object.
Availability	Local, full, and lightweight clients.
Syntax	*pingIntervals* = *sysObject*.*PingIntervals* **Element** — **Description** *sysObject* — The *System* object to be acted on. *pingIntervals* — The *PingIntervals* object to be returned.
Data Type	*PingIntervals* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

RecoveryStatus

Summary	Contains the *RecoveryStatus* object associated with the

	specified *System* object. The *RecoveryStatus* object provides status information on how a database recovery is proceeding. You can only access this property from a local client. You can read this property from an *OnSystemNssIdle* callback while the process that initiated the recovery is waiting for it to complete. Alternatively, another client, such as a plug-in application, can read it because plug-in applications always run as separate processes.
Availability	Local clients.
Syntax	*rsObject = systemObject.RecoveryStatus* **Element** — **Description** *rsObject* — The *RecoveryStatus* object to be returned. *systemObject* — The *System* object to be acted on.
Data Type	*RecoveryStatus* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

RegisterServicePin

Summary	Specifies whether registration occurs automatically when a service pin message is received. Registration causes the device to appear in the **Discovered.Uninstalled** subsystem.
Availability	Local, full, and lightweight clients.
Syntax	*regServPinFlag = systemObject.RegisterServicePin* **Element** — **Description** *regServPinFlag* — A Boolean value indicating whether whether registration occurs automatically when a service pin message is received. **TRUE**. Unregistered devices will automatically be registered and placed in the **Discovered**.**Uninstalled Subsystem** when a service pin message is received. **FALSE**. Devices will not automatically be registered. Devices will only be registered when their *Commission* method is

		invoked. This property defaults to True if the *InstallOptions* property was set to **lcaSharedMedia (2)** before the *System* was opened for the first time. Otherwise, it defaults to False.
	systemObject	The *System* object to be acted on.
Data Type	Boolean.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

RemoteChannel

Summary	If you are developing a Full client application, you can use this property to specify the Channel to which your application's *NetworkServiceDevice* is attached. When a Full client application opens a system, OpenLNS automatically determines the channel to which the application's *NetworkServiceDevice* is attached. However, OpenLNS cannot determine the correct channel if the channel is one of several channels connected by routers that are configured as repeaters or permanent bridges (the *Class* property is set to **lcaRepeater**, **lcaPermanentRepeater** or **lcaPermanentBridge**). In this scenario, you can set the *RemoteChannel* property to specify the channel to which the *NetworkServiceDevice* is attached. You must set this property before you open the *System* object. For more information on opening the *System* object, see the *OpenLNS Programmer's Guide*. You must explicity set this property before it can be read. If you attempt to read this property before it has been set, the **LCA, #150 lcaErrNotYetSet** exception will be thrown.	
Availability	Full client.	
Syntax	*channelObject= systemObject.***RemoteChannel**	
	Element	**Description**
	channelObject	The *Channel* on which the remote application resides.
	systemObject	The *System* object to be acted on.
Data Type	*Channel* object.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

RepeatTimer

Summary	Sets the repeat timer value that will be used on the system for network management messages only. You can set the number of repeat messages that will be sent by writing to the *RetryCount* property.
Availability	Local, full, and lightweight clients.

<table>
<tr><td>Syntax</td><td>repeatTimerValue = object.RepeatTimer

Element — Description

repeatTimerValue — The encoded value of the repeat timer.

The default value for this property is determined based on the network topology. It is recommended that you do not change this from the default value. If the default value is not suitable for your application, it is recommended that you use the Delay property of each Channel object on the network to ensure that each message is sent at the correct interval.

However, this property does accept a range of encoded values from 0 to 15. The encoded repeat timer values are as follows:

Encoded Value — Seconds
0 — 0.016
1 — 0.024
2 — 0.032
3 — 0.048
4 — 0.064
5 — 0.096
6 — 0.128
7 — 0.192
8 — 0.256
9 — 0.384
10 — 0.512
11 — 0.768
12 — 1.024
13 — 1.536
14 — 2.048
15 — 3.072

You can also write the value 254 to the property at any time to restore it to the default.

If you assign this property a value outside the acceptable range, the NS, #29 lcaErrNsOutOfRange exception will be thrown.

object — The System object to be acted on.</td></tr>
<tr><td>Data Type</td><td>Integer.</td></tr>
<tr><td>Read/Write</td><td>Read/write.</td></tr>
<tr><td>Added to API</td><td>Prior to LNS Release 3.0.</td></tr>
</table>

ResourceLanguageId

Summary	Controls which language should be used when displaying descriptive type information stored in device resource files. This property can contain one or more language codes that identify the language that should be used when displaying type information stored in resource files. When you pass multiple language codes to this property, they must be comma-separated, as in the following example: "frc, enz, rus." The language files will be searched for in the order that the codes are supplied. For example, if this property is set to "frc,enz,rus", OpenLNS will first look for the .frc (French Canadian) language files, and then the .enz and .rus language files when it searches the resource files. If none of the values in the list are found, the default of "enu" (U.S. English) will be used. All language codes are three characters long. See the *LonMark Device Resource File Developer's Guide* for information on resource language IDs and a partial list of the language codes you can pass to this property. Setting this property in the *ObjectServer* object overrides the value in the *System* object. It also allows this property to be set when the *System* object is not available (i.e. when performing standalone monitor and control). The default value for this property is "enu" (U.S. English). You can only access this property after you have *opened* the Object Server. **Note**: The *FormatLocale* object contains a *LanguageId* property. This property determines what language file OpenLNS will use to display data stored in the *FormattedValue* properties of *DataPoint* objects when the *FormatLocale* is being used by an application. See the *FormatLocale* object for more information on this.
Availability	Local, full, lightweight, and independent clients.
Syntax	*languageId = systemObject.ResourceLanguageId* **Element** — **Description** *languageId* — A comma-separated list of one or more language codes identifying the languages that should be used when displaying type information. *systemObject* — The *System* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

RetryCount

Summary	Sets the retry count used by the Object Server (and connections that use the default *System* value) for all network management messages that use the request/response, unacknowledged/repeat, or acknowledged message service types. You can set the interval for retry messages sent using the acknowledged or request messages by writing to the *TxTimer* property. You can set the interval for repeat messages sent using the unacknowledged/repeat message service by writing to the *RepeatTimer* property.
Availability	Local, full, and lightweight clients.
Syntax	*retryCountValue* = *object.RetryCount* **Element** — **Description** *retryCountValue* — The retry count value. This property has a range of 0–15. *object* — The *System* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

SecurityLevel

Summary	Controls the level of system key security. Specifically, this property controls the behavior of the Object Server when installing and updating devices with the *AuthenticationEnabled* property is set to True.
Availability	Local, full, and lightweight clients.
Syntax	*securityLevelValue* = *systemObject.SecurityLevel* **Element** — **Description** *securityLevelValue* — The level of security in the system. The valid values for this property, which are contained in the *ConstSecurityLevels* constant, are as follows: **0 lcaSecurityLevelKeyDistributionEnabled** Keys are distributed over the network whenever the *AuthenticationEnabled* property of a device is set to TRUE or a device with its *AuthenticationEnabled* property set to

		TRUE is commissioned or replaced. **1 lcaSecurityLevelKeyDistribution Disabled** Keys are never distributed over the network. For all devices that have their *AuthenticationEnabled* property set to TRUE, the installer is responsible for installing the authentication key. The key installed must match the value of the *System* object's *AuthenticationKey* property. OpenLNS will report the **NS, #4031 lcaErrNsUpdateFuncError** whenver it tries to update the device, and the *CommissionStatus* of the device will be set to **lcaCommissionUpdatesFailed (2),** until the key has been successfully installed into the device.
	systemObject	The *System* object to be acted on.
Data Type	Integer.	
Read/Write	Read/write.	
Added to API	Prior to LNS Release 3.0.	

ServiceStatus

Summary	Contains the *ServiceStatus* object for this *System* object. The *ServiceStatus* object contains information regarding the service that the client is currently peforming (in progress, queued, and so on) on the system. If your client application is taking a long period of time to access a property or invoke a method, you can access this property from an *OnSystemNssIdle* callback to check on the operation's status.	
Availability	Local, full, and lightweight clients.	
Syntax	*serviceStatusObject* = *systemObject.ServiceStatus*	
	Element	**Description**
	serviceStatusObject	The *ServiceStatus* object to be returned.
	systemObject	The *System* object to be acted on.
Data Type	*ServiceStatus* object.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

State

Summary	Describes the state of the system.
Availability	Local, full, and lightweight clients.
Syntax	*stateValue* = *object*.*State* **Element** — **Description** *stateValue* — The state of the system. The enumerated values for this property, which are stored in the *ConstDeviceStates* constant, are as follows: **Note:** The only two values that may be written to this property are **lcaStateCnfgOnline**(4) and **lcaStateSoftOffline**(12). All other properties are read-only. **2 lcaStateUncnfg** The application is loaded but the configuration is either not loaded, being reloaded, or deemed corrupted due to a configuration checksum error. A Neuron Chip also can make itself unconfigured by calling the Neuron C function **go_unconfigured()**. The device's service LED flashes at a one second rate in this state. **3 lcaStateNoApplUncnfg** No application is loaded yet, the application is in the process of being loaded, or the application has been deemed corrupted due to an application checksum error or signature inconsistency. The application does not run in this state. The device's service LED is steadily on in this state. **4 lcaStateCnfgOnline** Normal device state. The application is running and the configuration is considered valid. This is the only state in which messages addressed to the application are received. In all other states, they are discarded. The device's service LED is off in this state. **6 lcaStateCnfgOffline**

		Application loaded but not running. The configuration is considered valid in this state; the network management authentication bit is honored. The device's service LED is off in this state. **12 lcaStateSoftOffline** The device has an application, is configured, and is soft-offline. It will go online when it is reset or when requested to go online. The device's service LED is off in this state. **140 lcaStateCnfgBypass** The application confirmed the offline request, but is still running (bypass mode). The device's service LED is off in this state.
	object	The router to be acted on.
Data Type	Integer.	
Read/Write	Read /write.	
Added to API	Prior to LNS Release 3.0.	

Subnets

Summary	Contains the *Subnets* collection object associated with the specified *System*. The *Subnets* collection contains subnets that are associated with *object*. For example, the *Subnets* property returns a collection of all the *Subnet* objects in the system.	
Availability	Local, full, and lightweight clients.	
Syntax	*subnetCollection* = *object*.*Subnets*	
	Element	**Description**
	subnetCollection	The returned *Subnets* collection
	object	The *System* object to be acted upon.
Data Type	*Subnets* collection object.	
Read/Write	Read only.	
Added to API	Prior to LNS Release 3.0.	

Subsystems

Summary	Contains the *Subsystems* collection object associated with the specified *System*. A *Subsystem* object can in turn contain a collection of *Subsystems*. A *System* object's *Subsystems* collection contains two default *Subsystems* upon creation. These *Subsystems* are named "ALL", which lists all of the devices in the system and "Discovered", which lists all devices discovered by the object server that have not yet been associated with a subsystem. This includes both unconfigured devices discovered by the NSS and configured devices that were added by some other network management application that does not use the Object Server. The *System* object's *Subsystems* collection contains the top-level subsystems in the user's subsystem hierarchy plus two pre-defined subsystems: the ALL and the Discovered subsystem. • The **ALL** subsystem lists all of the devices in the system. • The **Discovered** subsystem lists all devices discovered by the object server that have not yet been associated with a subsystem. This includes both unconfigured devices discovered by the NSS and configured devices that were added by some other network management application that does not use the Object Server.
Availability	Local, full, and lightweight clients.
Syntax	*subsystemCollection = object.Subsystems* **Element** — **Description** *subsystemCollection* — *Subsystems* collection associated with the *object*. *object* — The *System* object.
Data Type	*Subsystems* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TemplateLibrary

Summary	Contains the *TemplateLibrary* object associated with the specified *System* object. The *TemplateLibrary* object contains all of the collections of the various templates known to the *ObjectServer*.
Availability	Local, full, and lightweight clients.
Syntax	*libraryObject = systemObject.TemplateLibrary* **Element** — **Description** *libraryObject* — Returned *TemplateLibrary* object.

	systemObject The *System* object to be acted on.
Data Type	*TemplateLibrary* object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

TxTimer

Summary	Sets the value of the transmit timer used for network management messages. You can use this property to specify the interval between between retries for acknowledged (**lcaSvcAckd)** and request (**lcaSvcRequest**) messages sent during network management operations. LNS retries message transmissions when no acknowledgment of (or response to) an original message is received This interval applies to network management messages only. To set this interval for monitor and control messages for temporary monitor points, use the *DsTxTimer* property. To set this interval for monitor and control messages for permanent monitor points, use the *ConnectDescTemplate* object's *TransmitTimer* property for connections involving those monitor points.
Availability	Local, full, and lightweight clients.
Syntax	*timerValue* = *systemObject.TxTimer* **Element** — **Description** *repeatTimerValue* — The encoded value of the repeat timer. The default value for this property is determined based on the network topology. It is recommended that you do not change this from the default value. If the default value is not suitable for your application, it is recommended that you use the *Delay* property of each *Channel* object on the network to ensure that each message is sent at the correct interval. However, this property does accept a range of encoded values from 0 to 15. The encoded repeat timer values are as follows: **Encoded Value** — **Seconds** 0 — 0.016 1 — 0.024 2 — 0.032 3 — 0.048 4 — 0.064

	5 0.096 6 0.128 7 0.192 8 0.256 9 0.384 10 0.512 11 0.768 12 1.024 13 1.536 14 2.048 15 3.072 You can also write the value 254 to the property at any time to restore it to the default. *timerValue* The encoded value of the transmit timer. *systemObject* The *System* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

UncommissionedDeviceCount

Summary	Returns the number of uncommissioned devices discovered by network recovery. At the completion of network recovery, devices are grouped into three categories. • Installed devices that were recovered are counted by the *CommissionedDeviceCount* property. • Installed devices that could not be recovered, and had to be uninstalled, are counted by the *UninstalledDeviceCount* property. • Devices that were discovered by network recovery but had not previously been installed are counted by the *UncommissionedDeviceCount* property.
Availability	Local, full, and lightweight clients.
Syntax	*numDevices = systemObject.UncommissionedDeviceCount* **Element** **Description** *numDevices* The number of uncommissioned devices discovered by the network recovery process. *systemObject* The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

UninstalledDeviceCount

Summary	Returns the number of devices uninstalled by network recovery. At the completion of network recovery, devices are grouped into three categories. • Installed devices that were recovered are counted by the *CommissionedDeviceCount* property. • Installed devices that could not be recovered, and had to be uninstalled, are counted by the *UninstalledDeviceCount* property. • Devices that were discovered by network recovery but had not previously been installed are counted by the *UncommissionedDeviceCount* property.
Availability	Local, full, and lightweight clients.
Syntax	*numDevices = systemObject.UninstalledDeviceCount* **Element** — **Description** *numDevices* — The number of devices uninstalled by network recovery. *systemObject* — The *System* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

UpdateInterval

Summary	The rate (in seconds) at which the background database device image reconciliation process runs after a device update failure occurs on the network. A device update failure occurs when a transaction has been completed and committed to the OpenLNS database, but OpenLNS is unable to load the information into the physical device because of some error. When OpenLNS is unable to update a device upon committing a transaction, it retries updating the devices at the interval defined by this property. The specified interval is dependent on the setting of the *network management mode*: • If the *network management mode* is set to **lcaMgmtModeDeferConfigUpdates (1)**, only update failures that occurred while the network management mode was set to **lcaMgmtModeDeferConfigUpdates (1)** will be retried. Update failures that occurred while the network management mode was set to **lcaMgmtModePropagateConfigUpdates (0)** will not be retried until the network management mode has been restored to the **lcaMgmtModePropagateConfigUpdates (0)** value. • If the *network management mode* is set to **lcaMgmtModePropagateConfigUpdates (0)**, all update failures will be retried at this interval. Device update failures are indicated by *NS exceptions* in the 4030–4089 range. You can force a retry of failed updates with the *RetryUpdates* method. If you are receiving persistent update failures for a device, you may need to re-commission the device. You can do this with the method.
Availability	Local, full, and lightweight clients.
Syntax	*updateIntervalValue* **=** *object*.**UpdateInterval** **Element** — **Description** *updateIntervalValue* — The update interval rate in seconds. A value of **0** indicates that the backround process is turned off. *systemObject* — The *System* object to be acted on.
Data Type	Long.
Read/Write	Read/write.
Added to API	Prior to LNS Release 3.0.

Systems

The *Systems* object represents a collection of *System* objects. Each *Systems* collection contains a single *System* object. The *System* object associated with a network has the same name as the network. The *Systems* object is automatically added when you add a new *Network* object.

The following table summarizes the *Systems* object.

Description	A collection of *System* objects.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*Network* object.
Default Property	*Item*.
Methods	None.
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *Systems* object does not contain any methods.

Properties

The *Systems* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Systems* object in the *ConstClassIds* constant:

	3 lcaClassIdSystem *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** **Description** *returnValue* The number of objects in the collection as a long integer. *object* The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns a *System* object from a *Systems* collection. You can retrieve a *System* object from its *Systems* collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. Index values start at 1. You can also retrieve a *System* object from a *Systems* collection with the *Name* property by passing the *System* object's name as a string expression.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** **Description** *retrievedObject* The *System* object retrieved from the collection. *collObject* The *Systems* collection object to be acted

	on. *index* — A Long type specifying the ordinal index of the *System* object to be retrieved. *stringExpression* — A string type specifying the name of the *System* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements.

	• If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

TemplateLibrary

The *TemplateLibrary* object contains all of the templates in a *System* object and represents a catalog where definitions that apply to multiple objects are stored. The following table summarizes the *TemplateLibrary* object.

Description	Contains all of the templates in a *System* object.
Added to API	Prior to LNS Release 3.0.
Accessed Through	*System* object.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *ConnectDescTemplates* • *Description* • *DeviceTemplates* • *Parent*

Methods

The *TemplateLibrary* object does not contain any methods.

Properties

The *TemplateLibrary* object contains the following properties:

- *ClassId*
- *ConnectDescTemplates*
- *Description*
- *DeviceTemplates*
- *Parent*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *TemplateLibrary* object in the *ConstClassIds* constant: **21 lcaClassIdTemplateLibrary** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

ConnectDescTemplates

Summary	Contains the *ConnectDescTemplates* collection object associated with the specified *TemplateLibrary* object. This is a collection of all *ConnectDescTemplate* objects known to the system.
Availability	Local, full, and lightweight clients.
Syntax	*cdtColl* = *tlObject*.*ConnectDescTemplates* **Element** — **Description** *cdtColl* — The *ConnectDescTemplates* collection to be returned. *tlObject* — The *TemplateLibrary* object to be acted on.
Data Type	*ConnectDescTemplates* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Description

Summary	Stores description information about the *TemplateLibrary* object.

Availability	Local, full, lightweight, and independent clients. Note that some collections are not available to Independent clients.
Syntax	*stringValue* = *object*.**Description** **Element** — **Description** *stringValue* — A string description of the *TemplateLibrary* object. *object* — The *TemplateLibrary* object to be acted on.
Data Type	String.
Read/Write	Read/write.
Added to API	This depends on the object used to access the property. Generally, this property exists for the objects listed above as soon as they are added to the API. See the applicable object to determine what release it was introduced in.

DeviceTemplates

Summary	Contains the *DeviceTemplates* collection object associated with the specified *TemplateLibrary* object. This is a collection of all *DeviceTemplate* objects known to the system.
Availability	Local, full, and lightweight clients.
Syntax	*devTempColl* = *tlObject*.**DeviceTemplates** **Element** — **Description** *devTempColl* — The *DeviceTemplates* collection to be returned. *tlObject* — The *TemplateLibrary* object to be acted on.
Data Type	*DeviceTemplates* collection object.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the

		specified *object*.
	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

TestInfo

The *TestInfo* object contains information describing the results of the last test on the *AppDevice* or *RoutersSide* object. You can initiate a test on an *AppDevice* or *Router* by calling the *Test* method on it. The following table summarizes the *TestInfo* object.

Description	Contains information on device and router tests.
Added to API	LNS Release 3.0.
Accessed Through	*AppDevice* object. *Router* object.
Default Property	None.
Methods	None.
Properties	• *ActualDomainId* • *ActualNeuronId* • *ActualNodeId* • *ActualProgramId* • *ActualSubnetId* • *AuxResultData* • *ClassId* • *DetailInfo* • *ExpectedDomainId* • *ExpectedNeuronId* • *ExpectedNodeId* • *ExpectedProgramId* • *ExpectedSubnetId* • *IsDetailInfoValid* • *Parent* • *Status*

Methods

The *TestInfo* object does not contain any methods.

Properties

The *TestInfo* object contains the following properties:

- *ActualDomainId*
- *ActualNeuronId*
- *ActualNodeId*

- *ActualProgramId*
- *ActualSubnetId*
- *AuxResultData*
- *ClassId*
- *DetailInfo*
- *ExpectedDomainId*
- *ExpectedNeuronId*
- *ExpectedNodeId*
- *ExpectedProgramId*
- *ExpectedSubnetId*
- *IsDetailInfoValid*
- *Parent*
- *Status*

ActualDomainId

Summary	Contains the actual Domain ID of the tested device or router side if the *AuxResultData* is set to **LcaDomainIdData**. This property can be used in conjunction with the *AuxResultData* to analyze Domain ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information The domain ID is stored as a string of hexadecimal digits. For example, a 3-byte domain ID would be represented as follows: "32a0cf". Domain IDs can be 1, 3, or 6 bytes in length.
Availability	Local, full, and lightweight clients.
Syntax	*domainIdValue* = *toObject*.**ActualDomainId** **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *domainIdValue* — The actual domain ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ActualNeuronId

Summary	Contains the actual Neuron ID of the tested device or router side if the *AuxResultData* is set to **LcaNeuronIdData**. This property can be used in conjunction with the *ExpectedNeuronId* to analyze NeuronID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. Neuron IDs are stored as 12-digit hexadecimal strings (for example, "a327ff27ba44").

Availability	Local, full, and lightweight clients.
Syntax	*neuronIdValue* = *toObject*.**ActualNeuronId** **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *neuronIdValue* — The actual Neuron ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ActualNodeId

Summary	Contains the actual node ID of the tested device or router side if the *AuxResultData* is set to **LcaNodeData**. This property can be used in conjunction with the *ExpectedNodeId* to analyze node ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. The *NodeId* and *SubnetId* comprise the logical network address assigned to a device.
Availability	Local, full, and lightweight clients.
Syntax	*nodeIdValue* = *toObject*.**ActualNodeId** **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *nodeIdValue* — The actual node ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ActualProgramId

Summary	Contains the actual program ID of the tested device or router side if the *AuxResultData* is set to **LcaProgramIdData**. This property can be used in conjunction with the *ExpectedProgramId* to analyze Program ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. Every LonMark compliant LONWORKS device uses a unique, 16 digit, hexadecimal standard program ID that uses the following format: FM:MM:MM:CC:CC:UU:TT:NN

Availability	Local, full, and lightweight clients.
Syntax	*progIdValue* = *toObject*.**ActualProgramId** **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *progIdValue* — The actual program ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ActualSubnetId

Summary	Contains the actual subnet ID of the tested device or router side if the *AuxResultData* is set to **LcaSubnetData**. This property can be used in conjunction with the *ExpectedSubnetId* to analyze Subnet ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. The *NodeId* and *SubnetId* comprise the logical network address assigned to a device.
Availability	Local, full, and lightweight clients.
Syntax	*subnetIdValue* = *toObject*.**ActualSubnetId** **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *nodeIdValue* — The actual subnet ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

AuxResultData

Summary	Contains auxiliary information about the *TestInfo* object.
Availability	Local, full, and lightweight clients.
Syntax	*auxResultValue* = *toObject*.*AuxResultData* **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *auxResultValue* — The auxiliary information. The possible values for this element, which are contained in the *ConstTestResultAuxData* constant,

	are as follows: **0 lcaNoAuxData** There is no useful information in the Actual/Expected properties of the *LastTestInfo* object. **1 lcaNeuronIdData** The *ActualNeuronId* and *ExpectedNeuronId* properties contain information about a Neuron ID discrepancy. **2 lcaDomainIdData** The *ActualDomainId* and *ExpectedDomainId* properties contain information about a Domain ID discrepancy. **3 lcaSubnetNodeData** The *ActualNodeId*, *ExpectedNodeId*, *ActualSubnetId*, and *ExpectedSubnetId* properties contain information about a Subnet/Node ID discrepancy. **4 lcaProgramIdData** The *ActualProgramId* and *ExpectedProgramId* properties contain information about a Program ID discrepancy.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *TestInfo* object in the *ConstClassIds*

	constant: **71 lcaClassIdTestInfo** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

DetailInfo

Summary	Contains the *DetailInfo* object associated with the specified *AppDevice*, *RouterSide*, or *TestInfo* object. The *DetailInfo* object contains an error log and communications status information for an *AppDevice* or *RouterSide* object. If the *DetailInfo* property is accessed through a *TestInfo* object, the property will only contain valid information if the *IsDetailInfoValid* property is set to True.
Availability	Local, full, and lightweight clients.
Syntax	*detailInfoObject = object.DetailInfo* **Element** **Description** *detailInfoObject* The *DetailInfo* object associated with the device. *object* The *TestInfo* object from which to get status information.
Data Type	*DetailInfo* object.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ExpectedDomainId

Summary	Contains the expected Domain ID of the tested device or router side if the *AuxResultData* is set to **LcaDomainIdData**. This property can be used in conjunction with the *ActualDomainId* to analyze Domain ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information The domain ID is stored as a string of hexadecimal digits. For example, a 3-byte domain ID would be represented as follows: "32a0cf". Domain IDs can be 1, 3, or 6 bytes in length.

Availability	Local, full, and lightweight clients.
Syntax	*domainIdValue = toObject.AuxResultData* **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *domainIdValue* — The expected domain ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ExpectedNeuronId

Summary	Contains the expected Neuron ID of the tested device or router side if the *AuxResultData* is set to **LcaNeuronIdData**. This property can be used in conjunction with the *ActualNeuronId* to analyze NeuronID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. Neuron IDs are stored as 12-digit hexadecimal strings (for example, "a327ff27ba44").
Availability	Local, full, and lightweight clients.
Syntax	*neuronIdValue = toObject.ExpectedNeuronId* **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *neuronIdValue* — The expected Neuron ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ExpectedNodeId

Summary	Contains the expected node ID of the tested device or router side if the *AuxResultData* is set to **LcaNodeData**. This property can be used in conjunction with the to analyze Node ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. The *NodeId* and *SubnetId* comprise the logical network address assigned to a device.
Availability	Local, full, and lightweight clients.

Syntax	*nodeIdValue = toObject.ExpectedNodeId* **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *nodeIdValue* — The expected node ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ExpectedProgramId

Summary	Contains the expected program ID of the tested device or router side if the *AuxResultData* is set to **LcaProgramIdData**. This property can be used in conjunction with the *ActualProgramId* to analyze Program ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. Every LonMark compliant LONWORKS device uses a unique, 16-digit hexadecimal standard program ID that uses the following format: FM:MM:MM:CC:CC:UU:TT:NN.
Availability	Local, full, and lightweight clients.
Syntax	*progIdValue = toObject.ExpectedProgramId* **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *progIdValue* — The expected program ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ExpectedSubnetId

Summary	Contains the expected subnet ID of the tested device or router side if the *AuxResultData* is set to **LcaSubnetData**. This property can be used in conjunction with the *ActualSubnetId* to analyze Subnet ID discrepancies. If the *AuxResultData* property is set to some other value, this property contains no useful information. The *NodeId* and *SubnetId* comprise the logical network address assigned to a device.
Availability	Local, full, and lightweight clients.

Syntax	*subnetIdValue* = *toObject*.**ExpectedSubnetId** **Element** **Description** *toObject* The *TestInfo* object to be acted on. *nodeIdValue* The expected subnet ID of the tested device or router side.
Data Type	String.
Read/Write	Read only.
Added to API	LNS Release 3.0.

IsDetailInfoValid

Summary	Indicates whether the *TestInfo* object's *DetailInfo* property contains valid information.
Availability	Local, full, and lightweight clients.
Syntax	*validDetailValue* = *tiObject*.*IsDetailInfoValid* **Element** **Description** *validDetailValue* A Boolean value indicating whether the *DetailInfo* property contains valid information. **TRUE**. The *DetailInfo* property contains valid information. **FALSE**. The *DetailInfo* property contains invalid information. *tiObject* The *TestInfo* object to be acted on.
Data Type	Boolean.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** **Description** *parentObject* The object that is the parent of the

	specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Status

Summary	Indicates the test status reported by the *Test* method.
Availability	Local, full, and lightweight clients.
Syntax	*statusValue = tiObject.Status* **Element** — **Description** *toObject* — The *TestInfo* object to be acted on. *statusValue* — The enumerated values for this element, which are contained in the *ConstTestResults* constant, are as follows: **0 lcaTestResultGood** The device passed all applicable tests. **1 lcaTestResultComm** The OpenLNS Server was unable to communicate with the device using either Neuron ID or its subnet/node addressing. The Network Service Device might not be attached to the network, the Network Service Device might be unconfigured, the target device might not be attached to the network, the target device may be powered off, or the target device may be faulty. **2 lcaTestResultCommNeuronId** The OpenLNS Server was unable to communicate with the device using Neuron ID addressing. Because the device has not been added or defined, the OpenLNS Server could not test the device using subnet/node ID addressing. The OpenLNS Server might not be attached to the network, the Network Service Device might be unconfigured, the target device might not be attached to the network, the target device may be powered off, or the target device may be faulty. **3 lcaTestResultCommNeuronIdVerified**

		The OpenLNS Server was able to communicate with the device using subnet/node addressing, but could not communicate with the device using Neuron ID addressing. However, through the use of subnet/node addressed messages, the OpenLNS Server has verified that the device contains the expected Neuron ID. This might be caused by intermittent device or channel failures. You may want to retry this method to see if the error persists. **4 lcaTestResultCommSnode** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using subnet/node addressing. The OpenLNS Server was unable to verify whether the device has been configured with the proper domain/subnet/node address. The device may have reconfigured itself, the device may have been reconfigured by another network management tool, or the device may be faulty. The device may be restored using the *Commission* method. **5 lcaTestResultSnodeVerified** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using subnet/node addressing. However, through the use of Neuron ID addressed messages, The OpenLNS Server has verified that the device contains the expected subnet/node address. This error can occur if the target device is in an unconfigured state. The device may need to be recomissioned or it may be in the middle of a two-phase move. It also might be caused by intermittent device or channel failures. You may want to retry this method to see if the error persists. **6 lcaTestResultDuplicateSnode** The OpenLNS Server was able to communicate with the device using Neuron ID addressing. However, when using subnet/node addressing, the responding device contains a different Neuron ID. This failure indicates that multiple devices are configured with the same domain/subnet/node address. This could be

		caused by attaching a new device to the network which was previously configured as part of another network or by two networks sharing the same media and domain ID. This condition could also occur if the unexpected device was supposed to be removed from the system, but the OpenLNS Server was unable to update the device's network image during a *Remove* method; meanwhile, the removed subnet/node address has been reused for the device being tested. **7 lcaTestResultMismatchDomain** The OpenLNS Server was able to communicate with the device using Neuron ID addressing but could not communicate with the device using subnet/node addressing. Using Neuron ID addressing, the OpenLNS Server found that the domain ID configured in the device does not match the database. This result will be returned if the device's domain address has not be configured or the device has become unconfigured due to a checksum error. **8 lcaTestResultMismatchNeuronId** The OpenLNS Server was unable to communicate with the device using Neuron ID addressing. However, the device that responded to the subnet/node addressed test message contains a different Neuron ID. It appears that the subnet/node address is configured in the responding device but not the tested device. One possible cause is that the tested device is supposed to replace the responding device, but the network images in both devices have not been updated yet. You can resolve this problem by physically removing the obsolete device. **9 lcaTestResultMismatchSnode** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using its subnet/node addressing. Through the use of Neuron ID addressed messages, the NSS has found that the subnet/node address configured in the device does not match the database. This result will be returned if the device's domain address has not been configured.

	10 lcaTestResultNoNeuronId The device has not been assigned a Neuron ID. **11 lcaTestResultMismatchProgramId** The OpenLNS Server was able to communicate with the device using both Neuron ID and subnet/node addressing. However, the device does not contain the expected program ID. The device's program ID may have been changed by its application program. Host devices can modify the program ID of their attached network interface. The device should be Removed and Added. **12 lcaTestResultCommSnodeNotVerified** The OpenLNS Server was able to communicate with the device using Neuron ID addressing, but could not communicate with the device using subnet/node addressing. The OpenLNS Server did not attempt to verify that the device has been configured with the proper domain/subnet/node address because the device is currently authenticated, and reading the address would result in transmitting the key over the network. The device may have reconfigured itself, the device may have been reconfigured by another network management tool, or the device may be faulty. The device may be restored using the Commission method. **13 lcaTestResultAuthEnabled** The OpenLNS Server was able to communicate with the device using both Neuron ID addressing and subnet/node addressing. However, the device has network management authentication *enabled* despite the fact that the device's *AuthenticationEnabled* property is set to FALSE. The device may have enabled network management authentication itself, the device may have been reconfigured by another network management tool, or the device may be faulty. It may be possible to restore the device using the Commission method. **14 lcaTestResultAuthDisabled** The OpenLNS Server was able to

		communicate with the device using both Neuron ID addressing and subnet/node addressing. However, the device has network management authentication *disabled* despite the fact that the device's *AuthenticationEnabled* property is set to TRUE. The device may have disabled network management authentication itself, the device may have been reconfigured by another network management tool, or the device may be faulty. The device may be restored using the Commission method. **15 lcaTestResultKeyMismatch** The OpenLNS Server was able to communicate with the device using both Neuron ID addressing and subnet/node addressing. The device has network management authentication enabled and the node's AuthentictionEnabled property is set to TRUE. However, the device does not contain the current system authentication key. The device may have changed its authentication key itself, the device may have been reconfigured by another network management tool, or the device may be faulty. It may be possible to restore the device using the Commission method. **16 lcaTestResultInterfaceFailure** The OpenLNS Server was unable to communicate with the OpenLNS network interface. The OpenLNS network interface may have become disconnected or faulty. Exit all OpenLNS applications and perform diagnostics on the OpenLNS network interface using the LONWORKS Interfaces Control Panel application. **17 lcaTestResultInterfaceNotOnline** The OpenLNS network interface that the OpenLNS Server is attempting to use is not Online. Recommission the NetworkServiceDevice of the System object by calling the `System.NetworkServiceDevice.AppDevice.Commission` method, and make sure that the State property of the NetworkServiceDevice object (NetworkServiceDevice.AppDevice.State) is set to lcaOnline. **18 lcaTestResultInterfaceConfigError**

	The OpenLNS network interface that the OpenLNS Server is attempting to use is not property configured. Recommission the NetworkServiceDevice of the System object by calling the `System.NetworkServiceDevice.AppDevice.Commission` method.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

TypeSpec

The *TypeSpec* object contains a group of properties that OpenLNS uses to identify the type a network variable should use. You can write new values to these properties to change the network variable's type if the network variable's *ChangeableTypeSupport* property is set to **lcaNvChangeableTypeSdOnly (1)** or **lcaNvChangeableTypeSCPT (2).**

To change a network variable's type, follow these steps:

1. Access the network variable's *TypeSpec* object through the *TypeSpec* property of the *NetworkVariable* object.
2. Set the program ID, scope, and name of the new type you want to use by writing to the *ProgramId*, *Scope*, and *TypeName* properties of the *TypeSpec* object.
3. Optionally, invoke the *Lookup* method on the *TypeSpec* object to make sure that the program ID, scope and name entered in step 2 reference a valid type.
4. Read the *IsComplete* property to make sure that the *TypeSpec* object is complete. This step is only necessary if you are creating a new network variable, or changing a network variable's type from a type that was received from another network variable.
5. Pass the modified *TypeSpec* object back to the *TypeSpec* property of the network variable. At this point, OpenLNS will use the values entered in step 2 to find the definition of the type in the resource files, and assign values to the *Index*, *Length*, and *ObjectType* properties of the *TypeSpec* object.
   - If OpenLNS is unable to find the resource file for the program ID entered in step 2, the **LCA #154 lcaErrUnavailableResourceFiles** exception will be thrown.
   - If OpenLNS finds the resource file but is unable to find the type name referenced in step 2, the **LCA #155 lcaErrNotFoundInResourceFiles** exception will be thrown. Make sure that the network variable can support the new type before assigning it.
   - If the length of the new type is too long for the network variable, the **LCA#156 lcaErrTypeLengthTooLong** exception will be thrown.

The following table summarizes the *TypeSpec* object.

Description	Contains a group of properties that OpenLNS uses to identify the type a network variable should use.
Added to API	LNS Release 3.20.
Accessed Through	*NetworkVariable* object.
Default Property	None.

Methods	• *Lookup*
Properties	• *ClassId* • *Index* • *IsComplete* • *Length* • *ObjectType* • *Parent* • *ProgramId* • *Scope* • *TypeName*

Methods

The *TypeSpec* object contains the following methods:

- *Lookup*

Lookup

Summary	Validates that the scope, program ID, and format name assigned to a *TypeSpec* object references a valid network variable type.
Availability	Local, full, and lightweight clients.
Syntax	*typeSpec*.**Lookup** **Element** — **Description** *typeSpec* — The *TypeSpec* object to be acted upon.
Added to API	LNS Release 3.20.

Properties

The *TypeSpec* object contains the following properties:

- *ClassId*
- *Index*
- *IsComplete*
- *Length*
- *ObjectType*
- *Parent*
- *ProgramId*
- *Scope*
- *TypeName*

ClassId

Summary	Identifies the object class of this object.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId**

	Element	**Description**
	classIdValue	The object class of the object. The following value is defined for the *TypeSpec* object in the *ConstClassIds* constant: **94 lcaClassIdTypeSpec**
	object	The object to be acted on.
Data Type	Integer.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

Index

Summary	Indicates the index number of the type referenced by this *TypeSpec* object, as defined in the resource file that contains the type's definition. When new values are written to the *Scope*, *ProgramId*, *TypeName* properties of the *TypeSpec* object and the *TypeSpec* object is passed back to the network variable, OpenLNS sets the *Index* property automatically. If the type referenced by the *TypeSpec* object is a standard type, this will be the SNVT index of the type. If the type is a user-defined type, this will be the index of the type as defined in the resource files containing the definition of the type.	
Availability	Local, full, and lightweight clients.	
Syntax	*indexValue* = *typeSpec.Index* **Element** **Description** *indexValue* The index number of the type. *typeSpec* The *TypeSpec* object to be acted on.	
Data Type	Long.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

IsComplete

Summary	Indicates whether the *TypeSpec* object is complete. Generally, you do not need to check whether a *TypeSpec* object is complete before passing it back to a network variable. However, if you are creating a new network variable or changing a network variable's type from a type that was received from another network variable, you need to check if it is complete. This property may be one of the following values:

	• **TRUE**. The *TypeSpec* object is complete. • **FALSE**. The *TypeSpec* object is incomplete. Several conditions may cause a *TypeSpec* object to be incomplete. For example, if you try to get a *TypeSpec* object for a network variable with a changeable UNVT type that uses a configuration property, and the category of the configuration property is either CAT_INITIAL or CAT_NULL, then the *TypeSpec* object would be incomplete.
Availability	Local, full, and lightweight clients.
Syntax	*complete* = *typeSpec.IsComplete* **Element** — **Description** *complete* — A Boolean value indicating whether the *TypeSpec* object is complete. *typeSpec* — The *TypeSpec* object to be acted on.
Data Type	Boolean.
Read/Write	Read only
Added to API	LNS Release 3.20.

Length

Summary	Indicates the length (in bytes) of network variables that use the type referenced by this *TypeSpec* object. When new values are written to the *Scope*, *ProgramID*, *TypeName* properties of the *TypeSpec* object and the *TypeSpec* object is passed back to the network variable, OpenLNS sets the *Length* property automatically. When you assign the *TypeSpec* object to a network variable, make sure that the value of this property does not exceed the value assigned to the network variable's *MaxLength* property. Otherwise, the **LCA#156 lcaErrTypeLengthTooLong** exception will be thrown.
Availability	Local, full, and lightweight clients.
Syntax	*lengthInBytes* = *typeSpec.Length* **Element** — **Description** *lengthInBytes* — The length of network variables (in bytes) using the type referenced by this *TypeSpec* object. *typeSpec* — The *TypeSpec* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

ObjectType

Summary	Indicates the type of objects to which the type should be applied. When new values are written to the *Scope*, *ProgramID*, *TypeName* properties of the *TypeSpec* object and the *TypeSpec* object is passed back to the network variable, OpenLNS sets the *ObjectType* property automatically.
Availability	Local, full, and lightweight clients.
Syntax	*objectType* = *typeSpec*.*ObjectType* **Element** — **Description** *objectType* — The objects to which the type applies. The possible values for this element, which are specified in the *ConstTypeSpecObjectType* constant, are as follows: **0 lcaTypeSpecNv** The type applies to network variables. **2 TypeSpecLmo** This type applies to LonMark objects or functional blocks (SFPTs and UFPTs). **2 TypeSpecFb** An alias for **TypeSpecLmo**. *typeSpec* — The *TypeSpec* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.20.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description**

	parentObject	The object that is the parent of the specified *object*.
	object	Any object for which the parent is desired.
Data Type	Object.	
Read/Write	Read only.	
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.	

ProgramId

Summary	Contains the program ID of the *TypeSpec* object. The program ID is stored as a unique 16-digit hexadecimal number in the following format: FM:MM:MM:CC:CC:UU:TT:NN. **Note**: You may not use colons when writing the program ID. For a device with a program ID of 80:00:01:01:28:80:04:02, you can write 8000010128800402 in this property. Each *TypeSpec* object references a type. OpenLNS uses the *ProgramId* property in conjunction with the *Scope* property of the applicable *TypeSpec* object to determine which resource file contains the type's definition. The type to use within that resource file is identified by the *TypeName* property. • If the *TypeSpec* object applies to a network variable that has a standard network variable type (SNVT), the *ProgramID* property is set to 0000000000000000. • If the *TypeSpec* object applies to a network variable that has a user-defined network variable type (UNVT), the *ProgramID* property is set to match the program ID of the device containing the network variable or data point.	
Availability	Local, full, and lightweight clients.	
Syntax	*programIdValue = typeSpec.ProgramID*	
	Element	**Description**
	programIdValue	The length of network variables (in bytes) using the type referenced by this *TypeSpec* object.
	typeSpec	The *TypeSpec* object to be acted on.
Data Type	String.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

Scope

Summary	Each *TypeSpec* object references a type defined in a LonMark resource file. This property indicates the scope of the resource file that contains that type's definition. The scope of a resource file determines which devices will use the types defined in that resource file.
Availability	Local, full, and lightweight clients.
Syntax	*scopeValue* = *typeSpec.Scope* **Element** — **Description** *scopeValue* — The scope of the device resource file containing the definition of this type. The possible values of this element, which are stored in the *ConstResourceScope* constant, are as follows: **0 lcaResourceScopeStandard** This scope applies to all devices. **1 lcaResourceScopeClass** This scope applies to all devices of a specified device class from any manufacturer. **2 lcaResourceScopeSubclass** This scope applies to all devices of a specified device class and device subclass from any manufacturer. **3 lcaResourceScopeMfg** This scope applies to all devices of a specified manufacturer. **4 lcaResourceScopeMfgClass** This scope applies to all devices of a specified manufacturer and device class. **5 lcaResourceScopeMfgSubClass** This scope applies to all devices of a specified manufacturer, device class and device subclass. **6 lcaResourceScopeMfgModel** This scope applies to all devices of a specified manufacturer, device class, device subclass and model. **-1 lcaResourceScopeUnknown** The scope of the resource file is not known, or could not be found. **-2**

		lcaResourceScopeAutoDeterminatio n This value applies to the *Mode* property of *LonMarkObject* objects only. Select this value to have LNS determine the value of the *Mode* property for the *LonMarkObject* automatically. If you select this value, OpenLNS will iterate through all the available resource files from most specific to most general (highest scope to lowest scope) until it finds the functional profile template resource file containing the *LonMarkObject* object's definition. It will then assign the proper value to the *Mode* property. If OpenLNS is unable to determine the proper scope value, it will set the *Mode* property to **lcaResourceScopeUnknown (-1).**
	typeSpec	The *TypeSpec* object to be acted on.
Data Type	Short.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

TypeName

Summary	Contains the name of the type referenced by the *TypeSpec* object. Each *TypeSpec* object is accessed through a network variable, and identifies the type used by that network variable. The resource file that contains the definition of the type is identified by the *Scope* and *ProgramID* properties of the *TypeSpec* object	
Availability	Local, full, and lightweight clients.	
Syntax	*typeName = typeSpec.TypeName*	
	Element	**Description**
	typeName	The name of the type.
	typeSpec	The *TypeSpec* object to be acted on.
Data Type	String.	
Read/Write	Read/write.	
Added to API	LNS Release 3.20.	

UpgradeInfo

Each time you call the *Upgrade* method to upgrade an *AppDevice* object's external interface file, an *UpgradeStatus* object is returned. The *UpgradeStatus* object contains properties describing the results of the upgrade. This includes the *UpgradeInfos* property, which contains a collection of *UpgradeInfo* objects. Each *UpgradeInfo* object represents a component of the old external interface file, which could be a LonMarkObject, a network variable, a configuration property, a static or dynamic message tag, a monitor set, or a monitor point. To determine the external interface component to which an *UpgradeInfo* object applies, read the *Class* and *FromIndex* properties of this object.

The other properties of the *UpgradeInfo* object contain information describing how the component it represents was affected by the upgrade.

- The *FromIndex* and *FromOwnerIndex* properties indicate the device index value assigned to the component in the old external interface file, and the *ToIndex* and *ToOwnerIndex* properties indicate the device index value assigned to the component in the upgraded external interface file.
- The *Status* property indicates whether or not the component was retained by the new external interface.
- The *Reason* property indicates why the component was retained or removed from the new external interface.

The following table summarizes the *UpgradeInfo* object.

Description	Describes the results of a device XIF upgrade.
Added to API	LNS Release 3.0.
Accessed Through	*UpgradeInfos* object.
Default Property	None.
Methods	None.
Properties	• *Class* • *ClassId* • *FromIndex* • *FromOwnerIndex Parent* • *Reason* • *Status* • *ToIndex* • *ToOwnerIndex*

Methods

The *UpgradeInfo* object does not contain any methods.

Properties

The *UpgradeInfo* object contains the following properties:

- *Class*
- *ClassId*
- *FromIndex*
- *FromOwnerIndex Parent*

- *Reason*
- *Status*
- *ToIndex*
- *ToOwnerIndex*

Class

Summary	Indicates the type of external interface component described by this *UpgradeInfo* object. You can use this property in conjunction with the *ToIndex* or *FromIndex* properties to determine the component in the new and old external interfaces that the *UpgradeInfo* object represents.
Availability	Local, full, and lightweight clients.
Syntax	*classValue* = *uiObject*.*Class* **Element** — **Description** *classValue* — An enumeration indicating the type of external interface component represented by the *UpgradeInfo* object. The possible values for this property, which are contained in the *ConstUpgrade Class* constant, are as follows: **1 lcaUgClsLmObj** A *LonMarkObject*. **2 lcaUgClsNetVar** A *NetworkVariable*. **3 lcaUgClsMsgTag** A static *MessageTag*. **4 lcaUgClsConfigProp** A *ConfigProperty* which is not associated with a network variable or LonMark object. **5 lcaUgClsLmObjConfigProp** A configuration property that is associated with a LonMark object. **6 lcaUgClsNetVarConfigProp** A configuration property that is associated with a network variable. **7 lcaUgClsMonSet** A *MonitorSet*. **8 lcaUgClsMonPoint** A *MsgMonitorPoint* or *NvMonitorPoint*.

	9 lcaUgClsDynamicMessageTag A dynamic *MessageTag*. *uiObject* The *UpgradeInfo* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Prior to LNS Release 3.0.

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** **Description** *classIdValue* The object class of the object. The following value is defined for the *Upgrade Info* object in the *ConstClassIds* constant: **87 lcaClassIdUpgradeInfo** *object* The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

FromIndex

Summary	Returns the device index value assigned to the external interface component before the upgrade. This is the component in the external interface file that is represented by this *UpgradeInfo* object. If the *Class* property is set to **lcaUgClsConfigProp (4)**, **lcaUgClsLmObjConfigProp (5)** or **lcaUgClsNetVarConfigProp (6)**, the *UpgradeInfo* obect represents a configuration property. In this case, the *FromIndex* property contains the handle assigned to the configuration property in the external interface file before the upgrade, and not the device index. The handle is relative to the object containing the

	configuration property. If the configuration property is contained by a *LonMarkObject* or network variable, the *FromOwnerIndex* property contains the device index assigned to the owner *LonMarkObject* or network variable in the upgraded external interface.
Availability	Local, full, and lightweight clients.
Syntax	*indexValue* = *uiObject.FromIndex* **Element** **Description** *indexValue* The device index value assigned to the external interface component before the upgrade. *uiObject* The *UpgradeInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

FromOwnerIndex

Summary	If the *UpgradeInfo* object represents a configuration property, this property returns the device index value assigned to the parent *LonMarkObject* or network variable in the external interface file before the upgrade. If the *UpgradeInfo* object does not represent a configuration property that is contained within a *LonMarkObject* or network variable, this property will return -1.
Availability	Local, full, and lightweight clients.
Syntax	*indexValue* = *uiObject.FromIndex* **Element** **Description** *indexValue* The device index value assigned to the external interface component before the upgrade. *uiObject* The *UpgradeInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Reason

Summary	Indicates the reason the external interface component represented by the *UpgradeInfo* object was retained or removed during the external interface upgrade. You can use the *Status* property to determine whether the external interface component represented by this property was removed or preserved in the new external interface.
Availability	Local, full, and lightweight clients.
Syntax	*reasonValue* = *uiObject*.*Reason* **Element** — **Description** *reasonValue* — An enumerated value indicating the reason this external interface component was preserved or deleted. The enumerated values for this element, which are contained in the *ConstUpgrade Reason* constant, are as follows: **1 lcaUgRsType** The external interface component was preserved because it matched the type of a component in the new external interface (it is the same type of component, has the same direction [if a network variable], the same size, and so on). External interface

		components can not be matched if their types differ. **2 lcaUgRsName** The external interface component was preserved because it matched the name of a component in the new external interface. **4 lcaUgRsIndex** The external interface component was preserved because it matched the index of a component in the new external interface. **8 lcaUgRsObjMembership** The external interface component was preserved because it is a network variable or configuration property which is a member of a LonMark object, and its member index and member number in the old LonMark object is the same in the new LonMark object (which may have a new object ID) in the new external interface. **16 lcaUgRsRecreated** The external interface component was re-created in the new external interface. This only applies to dynamic network variables, monitor sets, and monitor points. **32 lcaUgRsConvertFromStatic** The external interface component existed as a static component on the old device interface, but not the new one. As a result, it has been added to the new device interface as a dynamic component. All such components will have the same programmatic name as the original, unless the original was an array. In this case an index value will be appended to the name, or if the name does not fit, the name will be truncated. This may result in the assignment of duplicate names, but an OpenLNS application may change both the user name (*Name* property) and programmatic name (*ProgrammaticName* property) in this case. A new custom interface will be created by OpenLNS to house network variables and LonMarkObjects that are converted from static to dynamic in this fashion. The

		name used for these custom interfaces is "LcaUpgrade<n>", where <n> is a sequential number reflecting the number of upgrades that the device has gone through since LNS 3.20 was installed. For example, the 3rd time the device is upgraded, this interface would be named "LcaUpgrade3." Note that static components are converted to dynamic components only if the new interface supports dynamic LonMark Objects, and the old interface did not. **4096 lcaUgRsCapacity** The external interface component was deleted because it did not fit in the new external interface. This only applies to dynamic network variables, message tags, monitor sets, and monitor points which could not be mapped or recreated because the new external interface has a lower capacity. **8192 lcaUgRsNonExistent** The external interface component was deleted because no matching component could be found in the new external interface. **16384 lcaUgRsNoOwner** The monitor point was deleted because the network variable it was monitoring does not exist. The monitor set will still be re-created, even if its member monitor points could not be. **32768 lcaUgRsNoMember** The monitor point was deleted because the network variable it was monitoring was deleted, or the address table entry the monitor point was using was deleted.
	uiObject	The *UpgradeInfo* object to be acted on.
Data Type	Long.	
Read/Write	Read only.	
Added to API	LNS Release 3.0.	

Status

Summary	Indicates whether the external interface component represented by the *UpgradeInfo* object was preserved or deleted during the external interface upgrade. It also indicates whether any other modifictions were made as a result of the upgrade (for example, the component was moved). • If an external interface component is preserved, the new component will automatically have its properties set to match the old component, and the component will be bound into any connections of which the old component was a member. • If an external interface component is deleted, all connection information and properties set for that component will be erased. If the component is a configuration property, its value will also be erased. Use the *Reason* property to find out why a component was preserved or removed from the new interface. You can use the *Class* and *FromIndex* properties to determine the interface component to which the *Status* property applies. If the component is a configuration property contained by a *LonMarkObject* or *NetworkVariable*, you can use the *FromOwnerIndex* property to identify the *LonMarkObject* or *NetworkVariable* object.
Availability	Local, full, and lightweight clients.
Syntax	*statusValue = uiObject.Status* **Element** — **Description** *uiObject* — The *UpgradeInfo* object to be acted on. *statusValue* — The status of the component represented by the *UpgradeInfo* object. The enumerated values for this element, which are contained in the *ConstUpgrade Status* constant, are as follows: **1 lcaUgStsDeleted** The external interface component does not exist in the new external interface. The component will be removed from any connections and any configuration property values will be lost. **2 lcaUgStsPreserved** The external interface component was preserved in the new external interface. It will be a member of any connections in which the old component was a member and

	any configuration property values will be preserved. **4 lcaUgStsRelocated** The external interface component was preserved but relocated to a different index. See the *ToIndex* property and *ToOwnerIndex* property (if applicable) for the new location. **8 lcaUgStsNvAttribsMismatch** The network variable was preserved, but the polled or authenticated attributes do not match. The network variable will be removed from any bound connections and the **lcaUgStsDisconnNvAttrib** value will be set. This value only applies to network variables. **16 lcaUgStsDisconn** The network variable or message tag have been disconnected because it was removed. This value only applies to network variables and message tags. **32 lcaUgStsDisconnNvAttrib** The network variable is unbound because its polling or authentication attributes are different in the new interface. This value only applies to network variables. **64 lcaUgStsDisconnAliasOverflow** Some of the network variable's aliases are disconnected because they would not fit in the new interface. The network variable may remain in some connections. This value only applies to network variables. **128 lcaUgStsDisconnAddrOverflow** The network variable or message tag is removed from some or all connections because there are not enough address table entries in the new interface. This value only applies to network variables and message tags.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ToIndex

Summary	Returns the device index value assigned to the external interface component after the upgrade. This is the component in the external interface file that is represented by this *UpgradeInfo* object. If the *Class* property is set to **lcaUgClsConfigProp (4)**, **lcaUgClsLmObjConfigProp (5)** or **lcaUgClsNetVarConfigProp (6)**, the *UpgradeInfo* obect represents a configuration property. In this case, the *ToIndex* property contains the handle assigned to the configuration property in the external interface file in the upgraded external interface file, and not the device index. The handle is relative to the object containing the configuration property. If the configuration property is contained by a *LonMarkObject* or network variable, the *FromOwnerIndex* property contains the device index assigned to the owner *LonMarkObject* or network variable in the upgraded external interface. If the *UpgradeInfo* object's *Status* property is set to **lcaUgStsDeleted (1)**, then the component represented by this *UpgradeInfo* object was deleted from the external interface file during the upgrade. In this case, the *ToIndex* property will be **-1**.
Availability	Local, full, and lightweight clients.
Syntax	*indexValue* = *uiObject*.*ToIndex* **Element** — **Description** *indexValue* — The device index value assigned to the external interface component after the upgrade. *uiObject* — The *UpgradeInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

ToOwnerIndex

Summary	If the *UpgradeInfo* object represents a configuration property, this property returns the device index value assigned to the parent *LonMarkObject* or network variable in the external interface file after the upgrade. If the *UpgradeInfo* object does not represent a configuration property that is contained within a *LonMarkObject* or *NetworkVariable*, this property will return **-1**. If the *UpgradeInfo* object does represent a configuration

	property that is contained within a *LonMarkObject* or network variable, but the *Status* property is set to **lcaUgStsDeleted (1)**, the configuration proprty was deleted from the external interface file during the upgrade. In this case, this property will also return **-1**.
Availability	Local, full, and lightweight clients.
Syntax	*indexValue* = *uiObject*.*ToIndex* **Element** **Description** *indexValue* The device index value assigned to the external interface component after the upgrade. *uiObject* The *UpgradeInfo* object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	LNS Release 3.0.

UpgradeInfos

The *UpgradeInfos* object contains a collection of *UpgradeInfo* objects. The following table summarizes the *UpgradeInfos* object.

Description	A collection of *UpgradeInfo* objects.
Added to API	LNS Release 3.0.
Accessed Through	*UpgradeStatus* object.
Default Property	*Item*.
Methods	None.
Properties	• *ClassId* • *Count* • *Item* • *Parent* • *_NewEnum*

Methods

The *UpgradeInfos* object does not contain any methods.

Properties

The *UpgradeInfos* object contains the following properties:

- *ClassId*
- *Count*
- *Item*
- *Parent*
- *_NewEnum*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *Upgrade Infos* object in the *ConstClassIds* constant: **88 lcaClassIdUpgradeInfo** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Count

Summary	Returns the number of objects in a collection. You can use this property to enumerate a collection object.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*returnValue* = *object*.**Count** **Element** — **Description** *returnValue* — The number of objects in the collection as a long integer. *object* — The collection object to be acted on.
Data Type	Long.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Item

Summary	Returns an *UpgradeInfo* object from a *UpgradeInfos* collection. You can retrieve an *UpgradeInfo* object from its *UpgradeInfos*

	collection by passing its index (ordinal position) within that collection as the argument for the *Item* property. *Index* values start at 1.
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*retrievedObject* = *collObject*.**Item**(*index*) *retrievedObject* = *collObject*.**Item**(*stringExpression*) **Element** — **Description** *retrievedObject* — The *UpgradeInfo* object retrieved from the *UpgradeInfos* collection. *collObject* — The *UpgradeInfos* collection object to be acted on. *index* — A Long type specifying the ordinal index of the *UpgradeInfo* object to be retrieved. *stringExpression* — A string type specifying the name of the *UpgradeInfo* object to be retrieved.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is

	added to the API.

_NewEnum

Summary	Enables you to iterate over the objects in a collection using **For Each ... Next** statements. • If you are developing your OpenLNS app in Visual C++, you can browse a collection to find a particular item by using the *_NewEnum* property. See the **IEnumVARIANT** interface for details. • If you are developing your OpenLNS app in Visual Basic, you do not need to use the *_NewEnum* property because it is automatically used in the implementation of **For Each ... Next** statements. • If you are developing your OpenLNS app in C#, you do not need to use the *_NewEnum* property explicitly because it is automatically used in the implemention of **foreach** statements.
Availability	Local, full, and lightweight clients.
Syntax	*retrievedObject* = *collObject*.**_NewEnum** **Element** — **Description** *retrievedObject* — An expression that evaluates to the object retrieved. *collObject* — An iterator object that can be used to access members of the collection.
Data Type	Object.
Read/Write	Read only.
Added to API	OpenLNS.

UpgradeStatus

The *UpgradeStatus* object is returned by the *AppDevice* object's *Upgrade* method, and it contains a summary of all the changes made in the upgrade of the device's external interface. You can access the *UpgradeStatus* object returned the last time the *Upgrade* method was called on an *AppDevice* by reading the its *LastUpgradeStatus* property.

Description	Summarizes the changes made to a device's external interface during an upgrade.
Added to API	LNS Release 3.0.
Accessed Through	*AppDevice* object.
Default Property	None.
Methods	None.
Properties	• *ClassId* • *Parent* • *Result*

	• *Sequence* • *UpgradInfos*

Methods

The *UpgradeStatus* object does not contain any methods.

Properties

The *Upgrade Status* object contains the following properties:

- *ClassId*
- *Parent*
- *Result*
- *Sequence*
- *UpgradInfos*

ClassId

Summary	Identifies the object class of this object. This property allows the object type to be determined when it is unknown (for example, when the object was accessed using the *Parent* property of another object).
Availability	Local, full, lightweight, and independent clients. Note that some objects containing this property are not available to Independent clients.
Syntax	*classIdValue* = *object*.**ClassId** **Element** — **Description** *classIdValue* — The object class of the object. The following value is defined for the *UpgradeStatus* object in the *ConstClassIds* constant: **86 lcaClassIdUpgradeStatus** *object* — The object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Parent

Summary	Returns the object that spawned the current child object. The parent object is determined by the means in which a child object is accessed, not by the OpenLNS Object hierarchy.
Availability	Local, full, lightweight, and independent clients. Note that some collection objects are not available to Independent

	clients.
Syntax	*parentObject* = *object*.**Parent** **Element** — **Description** *parentObject* — The object that is the parent of the specified *object*. *object* — Any object for which the parent is desired.
Data Type	Object.
Read/Write	Read only.
Added to API	Depends on the object used to access the property. Generally, this property exists for an object as soon as the object is added to the API.

Result

Summary	Indicates if the upgrade was successful, failed as a result of a connection error, failed as a result of the node not being commissioned, or was not necessary. The *Upgrade* method is used to upgrade a device's external interface. The method returns an *UpgradeStatus* object, which contains information indicating whether the upgrade was successful, as well as information describing the changes made to each external interface component during the upgrade (for example, functional block, network variable, message tag, configuration property, monitor set, and monitor point).
Availability	Local, full, and lightweight clients.
Syntax	*resultValue* = *usObject*.**Result** **Element** — **Description** *resultValue* — The enumerated values for this element, which are contained in the *ConstUpgrade Result* constant, are as follows: **0 lcaUgResSuccess** The external interface upgrade was successful. The *UpgradeStatus* object's *UpgradeInfos* property contains the results of the upgrade on the individual external interface components. **1 lcaUgResNotRequired** The external interface upgrade failed because the new external interface specified has the same program ID and interface components as the old external interface.

	usObject The *UpgradeStatus* object to be acted on.
Data Type	Integer.
Read/Write	Read/write.
Added to API	LNS Release 3.0.

Sequence

Summary	Returns the number of times the external interface of the device containing the *UpgradeStatus* object has been upgraded since OpenLNS was installed on the client. If OpenOpenLNS has not been installed on the client, or if the device has not been upgraded since OpenLNS was installed, this property will return a value of **0**. The *Upgrade* method is used to upgrade a device's external interface. The method returns an *UpgradeStatus* object, which contains information indicating whether the upgrade was successful, as well as information describing the changes made to each external interface component during the upgrade (for example, functional block, network variable, message tag, configuration property, monitor set, and monitor point).
Availability	Local, full, and lightweight clients.
Syntax	*upgradeCount* = *upgradeStatus*.**Sequence** **Element** **Description** *upgradeCount* The number of times the device has been upgraded since OpenLNS was installed on the client. *upgradeStatus* The *UpgradeStatus* object to be acted on.
Data Type	Integer.
Read/Write	Read only.
Added to API	LNS Release 3.20.

UprgradeInfos

Summary	Contains a collection of *UpgradeInfo* objects. Each *UpgradeInfo* object contains information describing the changes made to an external interface component during the upgrade (for example, functional block, network variable, message tag, configuration property, monitor set, and monitor point).
Availability	Local, full, and lightweight clients.

Syntax	*ioColl* = *uiObject*.**UpgradeInfos** **Element** **Description** *uiColl* The *UpgradeInfos* collection to be returned. *uiObject* The *UpgradeInfo* object to be acted on.
Data Type	*UpgradeInfos* collection.
Read/Write	Read only.
Added to API	LNS Release 3.0.

3

Interfaces

This chapter details each interface in the OpenLNS Object hierarchy.

ILcaMsgMonitorPointListener

The *ILcaMsgMonitorPointListener* interface can be added to a COM object created in a multi-threaded programming environment such as Visual C++. You must then call the *Advise* method of the *MsgMonitorPoint* with the object that implements this interface as an argument. This will allow the object to receive updates from that message monitor point directly, rather than through the Object Server. The following table summarizes the *ILcaMsgMonitorPointListener* interface.

Description	Allows a COM object implementing this interface to directly receive updates from a message monitor point.
Added to API	LNS Release 3.0.
Methods	• *UpdateErrorEvent* • *UpdateEvent*

Methods

The *ILcaMsgMonitorPointListener* interface contains the following methods:

- *UpdateErrorEvent*
- *UpdateEvent*

UpdateErrorEvent

Summary	Allows an object to execute a callback for message monitor point and network variable update errors. This can only be used in multi-threading programming environments such as Visual C++. It cannot be used in Visual Basic. To create an object to receive *UpdateErrorEvent* updates, follow these steps: 1. Create an object that implements the *ILcaMsgMonitorPointListener* interface. 2. Define the object's behavior when the *UpdateErrorEvent* method is called. 3. Call the *Advise* method of the *MsgMonitorPoint* object with the object created in step 1 as the *object* element. OpenLNS will then provide event notification of update errors for the selected monitor point using callbacks instead of Windows messaging. You must call the *Advise* method from the event handler that is handling the update error events for the message monitor point. For more information on this, see the *Advise* method of the *MsgMonitorPoint* object. The object created in step 1 will now call this method whenever a message monitor point error is received.
Availability	Local and full clients.

Syntax	*object.UpdateErrorEvent updateType* **Element** **Description** *object* An object that implements the *ILcaMsgMonitorPointListener* interface. *updateType* A value indicating the type of monitor point update which returned an error. This will be set to 1.
Added to API	LNS Release 3.0.

UpdateEvent

Summary	Allows an object to execute a callback for message monitor point updates. This can only be used in multi-threading programming environments such as Visual C++. It cannot be used in Visual Basic. To cause events to be generated in this manner, follow these steps: 1. Create an object that implements the *ILcaMsgMonitorPointListener* interface. 2. Define the object's behavior when the *UpdateEvent* method is called. 3. Call the *Advise* method of the *MsgMonitorPoint* object with the object created in step 1 as the *object* element. OpenLNS will then provide event notification of update errors for the selected monitor point using callbacks instead of Windows messaging. You must call the *Advise* method from the event handler that is handling the update error events for the message monitor point. For more information on this, see the *Advise* method of the *MsgMonitorPoint* object. For completion code messages, the *InputDp*, *OutputDp*, and *Src* parameters are NULL.
Availability	Local and full clients.
Syntax	*UpdateErrorEvent*(*UpdateType* as Integer) **Element** **Description** *object* An object that implements the *ILcaMsgMonitorPointListener* interface. *updateType* A value indicating the type of message monitor point update. This will be set to **0**. *inputDp* The *DataPoint* object containing the message monitor point update. The *InputDp* parameter's *AutoRead* and *AutoWrite* properties are set to False. *outputDp* A *DataPoint* object that allows a response to be sent if the update is a

		request/response message. The *OutputDp* parameter's *AutoWrite* property is set to True; therefore, it will be sent as a response when you write to this *DataPoint* object.
	srcAddr	The *SourceAddress* object indicating the source device of the message monitor point update.
Added to API	LNS Release 3.0.	

ILcaNvMonitorPointListener

The *ILcaNvMonitorPointListener* interface can be added to a COM object created in a multi-threaded programming environment such as Visual C++. You must then call the *Advise* method of the *NvMonitorPoint* object with the object that implements this interface as an argument. This will allow the object to receive updates from that network variable monitor point directly, rather than through the Object Server.

The following table summarizes the *ILcaNvMonitorPointListener* interface.

Description	Allows a COM object implementing this interface to directly receive updates from a network variable monitor point.
Availability	Local and full clients.
Added to API	LNS Release 3.0.
Methods	• *UpdateErrorEvent* • *UpdateEvent*

Methods

The *ILcaNvMonitorPointListener* object contains the following methods:

- *UpdateErrorEvent*
- *UpdateEvent*

UpdateErrorEvent

Summary	Allows an object to execute a callback for network variable monitor point and network variable update errors. This can only be used in multi-threading programming environments such as Visual C++. It cannot be used in Visual Basic. To create an object to receive *UpdateErrorEvent* updates, follow these steps: 1. Create an object that implements the *ILcaNvMonitorPointListener* interface. 2. Define the object's behavior when the *UpdateErrorEvent* method is called. 3. Call the *Advise* method of the *NvMonitorPoint* object with the object created in step 1 as the *object* element. OpenLNS will then provide event notification of update errors for the selected monitor point using callbacks

	instead of Windows messaging. You must call the *Advise* method from the event handler that is handling the update error events for the network variable monitor point. For more information on this, see the *Advise* method of the *NvMonitorPoint* object. The object created in step 1 will now call this method whenever a network variable monitor point error is received.
Availability	Local and full clients.
Syntax	*object.UpdateErrorEvent updateType* **Element** — **Description** *object* — An object that implements either the *ILcaNvMonitorPointListener* interface. *updateType* — A value indicating the type of monitor point update which returned an error. This will be set to 1.
Added to API	LNS Release 3.0.

UpdateEvent

Summary	Allows an object to execute a callback for network variable monitor point updates. This can only be used in multi-threading programming environments such as Visual C++. It cannot be used in Visual Basic. To cause events to be generated in this manner, follow these steps: 1. Create an object that implements the *ILcaNvMonitorPointListener* interface. 2. Define the object's behavior when the *UpdateEvent* method is called. 3. Call the *Advise* method of the *NvMonitorPoint* object with the object created in step 1 as the *object* element. OpenLNS will then provide event notification of update events using callbacks instead of Windows messaging. You must call the *Advise* method from the event handler that is handling the update events for the network variable monitor point. For more information, see the *Advise* method of the *NvMonitorPoint* object. For completion code messages, the *InputDp*, *OutputDp*, and *Src* parameters are NULL.
Availability	Local and full clients.
Syntax	*UpdateErrorEvent*(*UpdateType* as Integer) **Element** — **Description** *object* — An object that implements the *ILcaNvMonitorPointListener* interface. *updateType* — A value indicating the type of network variable monitor point update. This will

		be set to **0**.
	inputDp	The *DataPoint* object containing the network variable monitor point update. The *InputDp* parameter's *AutoRead* and *AutoWrite* properties are set to False.
	outputDp	A *DataPoint* object that allows a response to be sent if the update is a request/response message. The *OutputDp* parameter's *AutoWrite* property is set to True; therefore, it will be sent as a response when you write to this *DataPoint* object.
	srcAddr	The *SourceAddress* object indicating the source device of the network variable monitor point update.
Added to API	LNS Release 3.0.	

ILcaProgressListener

The *ILcaProgressListener* interface can be added to an application to enable it to receive database validation updates directly through the interface, as opposed to through the Object Server. A database validation is initiated when the *Network.Validate* method is called.

If an object that implements the *ILcaProgressListener* interface is supplied to the function as the *progressCallback* argument, it enables the object to receive updates from that database validation process directly with the *ProgressUpdate* method. Otherwise, the update events from the process will need to be received through the Object Server via the *OnDbValidationEvent* event.

Note: If you are programming in Visual Basic, you should implement this as a _DLcaProgressListener interface. For Visual C++ and all other development environments, implement this as the *ILcaProgressListener* interface.

The following table summarizes the *ILcaProgressListener* interface.

Description	Allows an application to receive database validation updates directly through the interface.
Added to API	LNS Release 3.20.
Methods	• *ProgressUpdate*

Methods

The *ILcaProgressListener* object contains the following methods:

- *ProgressUpdate*

ProgressUpdate

Summary	Executes a callback to receive database validation update events through the *ILcaProgressListener* interface object. To create an object that receives database validation updates,

<table>
<tr><td></td><td>follow these steps:
1. Create an object that implements the ILcaProgressListener interface.
2. Define the behavior of the object when the ProgressUpdate method is called.
3. Call the Validate method of the Network with the object created in step 1 as the progressCallback argument. The object created in step 1 will now call this method whenever a database validation update event is received, and will be updated with the information delivered through the update event.</td></tr>
<tr><td>Availability</td><td>Local clients.</td></tr>
<tr><td>Syntax</td><td>object. ProgressUpdate(totalPercentage as Long, thisPhasePercentage as Long, thisPhaseNumber as Long, totalPhases as Long, thisPhaseName as String, thisStepName as String)
<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>totalPercentage</td><td>The percentage of the database validation that has been completed.
This element has a range between 0–100.</td></tr>
<tr><td>thisPhasePercentage</td><td>The percentage of the current phase of the database validation that has been completed.
You can use the thisPhaseName element to determine which phase of the database validation is currently being performed.
This element has a range between 0–100.</td></tr>
<tr><td>thisPhaseNumber</td><td>The number of the current phase.
This element has a range between 0–4,292,967,296.</td></tr>
<tr><td>totalPhases</td><td>The total number of phases to be performed during the database validation.
This element has a range between 0–4,292,967,296.</td></tr>
<tr><td>thisPhaseName</td><td>The name of the phase that is currently being performed.
The phase name will be returned as a string of up to 128 characters.</td></tr>
<tr><td>thisStepName</td><td>The step that is currently being performed.</td></tr>
</table></td></tr>
</table>

	The step name will be returned as a string of up to 128 characters.
Added to API	LNS Release 3.20.

4

OpenLNS Errors

This chapter lists and describes the errors that may be generated by the various OpenLNS components.

Errors

LNS errors may be produced by each of the LNS components. These are uniquely identified in the object server by the addition of a component offset value. Errors are therefore mapped into the following ranges:

Range	Error Type	Mapping
1000-16499	*Network Services Errors*	NS error 1 is LNS error 1001, NS error 2 is LNS error 1002, and so on.
16500-16599	*Network Interface Errors*	NI error 1 is LNS error 16501, NI error 16502 is LNS error 1002, and so on.
16600-16999	*Connection Errors*	CONNERR error 1 is LNS error 16601, CONNERR 2 is LNS error 16502, and so on.
17000-17999	*Object Server Errors*	LCA error 1 is LNS error 17001, LCA error 2 is LNS error 17002, and so on.
18000-18499	*Data Server Errors*	DS error 1 is LNS error 18001, DS error 2 is LNS error 18002, and so on.
18500-19999	Formatter errors	
20000-21999	Component errors	
22000-24999	User errors	

The following subsections describe the errors that may be generated by the following LNS components:

- Network Services Errors
- Network Interface Errors
- Connection Errors
- Object Server Errors
- Data Server Errors
- Formatter Errors
- VNI Errors

Network Services Errors

A network services error occurs when the OpenLNS Server is unable to successfully complete a service. These errors will appear in the format NS: #<Error Number>. The LNS values for these errors are equal to the error number plus 1000.

lcaErrNsCancel **1**

The service and the associated transaction were cancelled. This error occurs when an internal error occurs, such as the NSS engine restarting.

lcaErrNsNodeHasNoAppl **5**

The device was discovered to be applicationless, or not associated with a program. If you encounter this error, load the application into the device. For host-based nodes, ensure that the host application is running.

lcaErrNsBadManagerId **6**

The current manager ID does not correspond to a known manager. This is an internal error and should be reported to customer support.

lcaErrNsResourceProblem **7**

A problem occurred in resource allocation. This is an internal error and should be reported to customer support.

lcaErrNsInterfaceFailure **8**

The host was unable to communicate with the NSI. This may be due to problems communicating with the NSS engine, or with the network interface. If you encounter this error, ensure that the NSS engine process is running, and that the network interface is functioning properly.

lcaErrNsBadClient **9**

The specified server (or the current server) is undefined. This is an internal error and should be reported to customer support.

lcaErrNsBadServer **10**

The specified server (or the current server) is undefined.

lcaErrNsNoClient **11**

No client is currently defined. This is an internal error and should be reported to customer support.

lcaErrNsNoTxInProgress **12**

There is no transaction defined for the client. This warning can be expected when cancelling a method that is not part of an explicit transaction.

lcaErrNsNssNotFound **13**

The NSI has not been configured to communicate with the NSS. This error may occur when opening the database with the remote database collection if the clients NSD is not properly configured. To reconfigure the client's NSD, open the database remotely using the *Networks* collection.

lcaErrNsClientBlocked **14**

The client is already involved in a service request. You should invoke only one service per client. This includes all requests to start or commit transactions. When a client has already launched a service, wait until the current service completes before you invoke a new service. This is an internal error and should be reported to customer support.

lcaErrNsTxInProgress **15**

The client is already involved in a transaction. End the client's current transaction by calling CancelTransaction method, or wait for the transaction to complete and call the *StartTransaction* method, before starting a new transaction for that client.

You should avoid starting multiple transactions on a single network database within a single application.

lcaErrNsNotImplemented **16**

The specified service or option is not implemented by the specified server. This error could indicate a bad service parameter. This is an internal error and should be reported to customer support.

lcaErrNsErrorInfo **17**

The service failed. When you encounter this error, a service-specific error status is available in the LastError property of the *System* object.

lcaErrNsServiceFailure **18**

The requested service failed. This is an internal error, and should be reported to customer support. This error may occur for a remote client operation when the OpenLNS Server has shutdown.

lcaErrNsMsgRejectedByNode **19**

A node returned a negative response to a command from NSS. This could be due to a LonTalk authentication failure. It may also result from an inconsistency between the program interface defined in the database and what the node actually supports.

lcaErrNsPrematureRelease **20**

This is an internal error and should be reported to customer support.

lcaErrNsRootBusy **21**

A transaction is still in progress. This is an internal error and should be reported to customer support.

lcaErrNsRootDoesNotExist **22**

The service depends on a transaction that does not exist. This is an internal error and should be reported to customer support.

lcaErrNsNoFreeRootTransactions **23**

This is an internal error and should be reported to customer support.

lcaErrNsCantFindObject **24**

One of the objects needed to complete the service is not in the NSS's database.

lcaErrNsDuplicateObject **25**

An attempt was made to add an object that already exists.

lcaErrNsCommError **26**

The NSS cannot communicate with the node. This may be because the Neuron ID referenced by the application is invalid or does not exist, the NSS or the node is not physically attached to the media, the NSS is not configured (because the *MgmtMode* property was not set to lcaMgmtModePropagateConfigUpdates after the database was initially created), problems with the physical media, or the NSS timers (set using the *RetryCount* and *TxTimer* properties of the *System* object) are too low.

lcaErrNsOperationError **27**

The node failed to go online, offline or to reset when requested to do so. This may be due to a problem in the implementation of the device.

lcaErrNsUnsupportedInfo **28**

The requested information is not available from an application node. For example, transceiver status, SI/SD data, and network variable names are not always stored in the node.

lcaErrNsOutOfRange **29**

The value assigned to a property or parameter is either out of range, or invalid.

lcaErrNsEventFailure **30**

The generation of an event failed. This is an internal error and should be reported to customer support.

lcaErrNsDeferConfigUpdatesMgmntMode **31**

The requested service cannot be provided in the current management mode. When you encounter this error, set the system management mode to lcaMgmtModePropagateConfigUpdates (0), and try the operation again. You can change the system management mode by writing to the *MgmtMode* property of the *System* object.

lcaErrNsSubscriptionDblimit **32**

The NSS's event subscription table is full. This is an internal error, and should be reported to customer support.

lcaErrNsSequence **33**

A record from a binary external interface file or binary application image file was received out of sequence. If you encounter the error, the XFB or APB file may be corrupted.

lcaErrNsSegmentation **34**

Segmentation is not supported, or there is a problem with segment order. This is an internal error, and should be reported to customer support.

lcaErrNsChecksum **35**

A checksum error occurred while transferring the binary external interface or binary application image file to the NSS. If you encounter the error, the XFB or APB file may be corrupted.

lcaErrNsSessionError **36**

Invalid session handle or a session error. Make sure that sessions are begun and ended in pairs, and are always part of an explicit transaction.

lcaErrNsNoNeuronId **37**

An operation requiring a Neuron ID was attempted on a device with no Neuron ID. For example, if you invoke the Wink method on a device whose Neuron ID has not been set in the OpenLNS database, this exception will be thrown. In some other cases, this exception may be thrown if a device's Neuron ID has been set in the OpenLNS database, but the device has not been commissioned. For example, if you read the SelfDocumentation property of a device before that device has been commissioned, this exception will be thrown, regardless of whether or not the device's Neuron ID is set.

lcaErrNsProgramidMismatch **38**

Program IDs do not match. This may occur if there is an incorrect or out-of-date program version.

lcaErrNsObjectInUse **39**

A requested operation can't complete because the object involved in the operation is in use. For example, you may not be able to delete a node because that node is still included in connections.

lcaErrNsNodeHasNoSiData **40**

The node has no SI (self-information) data, when required.

lcaErrNsDbError **41**

An internal database error occurred. This is an internal error, and should be reported to customer support.

lcaErrNsOutOfJournal **42**

No more memory is available for journaling. This is an internal error, and should be reported to customer support.

lcaErrNsOutOfMemory **43**

No more memory is available for allocation.

lcaErrNsBadParms **44**

Invalid parameter values.

lcaErrNsNetworkHasInstalledNodes **46**

Attempt to add the NSS failed because application nodes have already been defined or added. You must install the NSS before the other nodes. This is an internal error, and should be reported to customer support.

lcaErrNsDbImportExport **47**

Requested operation can't proceed because a database import or export is currently in progress.

lcaErrNsProgramNotFound **48**

Could not find the program definition required to execute the service.

lcaErrNsIntegrityError **49**

Internal integrity error. This is an internal error, and should be reported to customer support.

lcaErrNsDblimit **50**

An unspecified database limit was exceeded.

lcaErrNsNodeNotFound **51**

Cannot find the node**.**

lcaErrNsDuplicateNode **52**

An attempt was made to add a node that is already defined.

lcaErrNsNodeDbLimit **53**

The node limit has been exceeded.

lcaErrNsDuplicateProgram **54**

The program template is already defined.

lcaErrNsProgramDblimit **55**

This is an internal error, and should be reported to customer support.

lcaErrNsNvmtNotFound **56**

The network variable or message tag cannot be found.

lcaErrNsNvmtDbLimit **57**

This is an internal error, and should be reported to customer support.

lcaErrNsChannelNotFound **58**

The channel cannot be found.

lcaErrNsProgramIntfMismatch **59**

The new program interface does not match the previously defined program interface; i.e., the number of network variables, message tags, or the structure of self identification data is inconsistent even though the program IDs match. This error may also be thrown if the SelfDocConsistency property of a *DeviceTemplate* object is set to a value that conflicts with the configurations of the devices using that template.

lcaErrNsProgramIntfUnsupported **60**

The program interface does not support the requested action.

lcaErrNsMsgError **61**

There was a failure in the messaging sub-system. This is an internal error, and should be reported to customer support.

lcaErrNsCancelError **62**

An error occurred while canceling a transaction or service. This is an internal error, and should be reported to customer support.

lcaErrNsNothingToCancel **63**

There was no service or transaction to cancel.

lcaErrNsNodeNotInstalled **64**

An operation on a node was requested that requires the node to be configured, but the node is not configured. When you encounter this error, make sure that the device in question has been configured.

lcaErrNsAuthViolation **65**

Authentication rules were violoated.

lcaErrNsChannelDblimit **66**

The network channel limit (currently 1000) has been reached.

lcaErrNsSubnetDblimit **67**

The network subnet limit (currently 255) has been reached.

lcaErrNsSubnetNotFound **68**

The specified subnet ID was not found. *Subnet* IDs must be created automatically by the NSS or defined manually using the Add method of the *Subnets* collection object.

lcaErrNsRouterDblimit **69**

The network router limit (currently 32766) has been reached.

lcaErrNsRouterNotFound **70**

The specified router handle does not exist.

lcaErrNsTimerRange **71**

The timer value computed for a connection or for the OpenLNS Object Server to communicate with a node exceeds the range supported by LONWORKS devices. This is an internal error, and should be reported to customer support.

lcaErrNsWrongChannel **72**

An attempt was made to add, commission, move, or replace a device that is on the wrong channel. A device is considered to be on the wrong channel when a configured, learning, or non-permanent bridge class router exists between the device and the channel on which it is to be placed. When you encounter this error, place the device or router on the correct channel, or define the device or router without a channel and have LNS determine the channel automatically.

lcaErrNsInsufficientRouters **73**

An attempt was made to create a connection between nodes on different channels, but no routers were available to complete a logical path between the channels. This error can occur when attempting to remove or move a router which has connections across it, or a router which connects an NSI to either the NSS or another NSI.

lcaErrNsNoSessionInProgress **74**

The *EndSession* method of the *System* object was called before a corresponding *BeginSession* was called. Make sure that a session is open before you call *EndSession*.

lcaErrNsTopologyPhysicalLoop **75**

The attempted router operation would have created a loop in the physical network topology.

lcaErrNsTopologyLogicalLoop **76**

The attempted router operation would have created a loop in the logical network topology. Normally, loops are initially detected as physical loops. However, it is possible to add a permanent repeater or bridge, so that a logical loop exists even though a physical loop does not.

lcaErrNsTopologySubnetViolation **77**

An attempt was made to define a device with a subnet that is incompatible with the devices channel, due to router constraints. To resolve this error, either leave the subnet undefined (so OpenLNS will define it) when you define the device, or make sure that the selected subnet is not in use on other logical channels.

lcaErrNsRouterMustBePermanent **78**

An attempt was made to illegally add a non-permanent router, or to change the class of a permanent router to non-permanent. Once a permanent router has been added to the topology, it cannot be changed to a non-permanent router, even by removing and then re-adding it.

lcaErrNsCantDetermineChannel **79**

An attempt was made to register or add a device without a specified channel handle, and the system was not able to automatically determine the channel on which the device resides. This error will occur if the channel the device resides on is connected to any

other channel by repeaters or permanent bridges. Also, this error will occur if the channel is not currently configured in the system.

lcaErrNsRedundantRoutersMustBeConfigured 80

An attempt was made to add a redundant router that was not of the configured class, to change the class of a redundant router to the non-configured class, or to add a redundant router to an existing non-redundant, non-configured class router.

lcaErrNsCantDetermineXcvrId 81

A channel was defined with a wildcard transceiver ID, but the transceiver ID could not be automatically determined by the system.

lcaErrNsRouterCantBePermanent 82

An attempt was made to change a non-permanent router to a permanent router by writing to the *Class* property of the Router object. A router can be changed in this way only by being moved.

lcaErrNsObjectLocked 83

An attempt was made to change a locked object. This is an internal error, and should be reported to customer support.

lcaErrNsInvalidContext 84

An attempt was made to invoke a service from an event handler that was called directly from a background task. This is an internal error, and should be reported to customer support.

lcaErrNsServerNotFound 85

The specified server could not be found. The server ID may be invalid. This is an internal error, and should be reported to customer support.

lcaErrNsNodeStateError 86

The node was in the wrong state for the attempted operation. To resolve this error, try setting the *State* property of the node's *AppDevice* object to lcaStateCnfgOnline.

lcaErrNsLmobjNotFound 87

The specified LonMark object could not be found.

lcaErrNsLmobjDbLimit 88

The *LonMarkObject* property database limit has been exceeded.

lcaErrNsInvalidSidata 89

The self-documentation information was invalid (e.g., improper LonMark definitions).

lcaErrNsServiceCommError 90

A communications error occurred between the client and the server.

lcaErrNsRequestNotAllowed 91

The requested service has been disabled. This error is usually the result of a remote full client performing an operation that has been disabled by the server.

lcaErrNsCpNotFound 92

The specified configuration property does not exist.

lcaErrNsCpDbLimit **93**

The configuration property database limit has been exceeded.

lcaErrNsCpRangeNotFound **94**

The specified configuration property range does not exist.

lcaErrNsCpRangeDblimit **95**

The configuration property range database limit has been exceeded.

lcaErrNsConnectionError **96**

Connection rebinding failed for a set of moved nodes. Possible causes are: no more address table slots, no more groups, or broadcast violation.

lcaErrNsFileLookupError **97**

An error occurred during a file operation, probably due to an out of range file index.

lcaErrNsFileIoError **98**

The file operation target failed to read/write a file.

lcaErrNsFileTimeoutError **99**

The target timed out during a file transfer operation.

lcaErrNsFileWindowError **100**

The target received a packet out of order during a file transfer operation.

lcaErrNsFileAuthError **101**

File transfer failed due to incorrect authentication.

lcaErrNsFileAccessUnavail **102**

Random file access is not implemented on the target.

lcaErrNsFileOpenFailure **103**

The target failed an open file operation.

lcaErrNsFileSeekInvalid **104**

The target failed a file seek operation.

lcaErrNsCantModifyConstCp **105**

An attempt was made to modify a constant configuration property.

lcaErrNsCpDefaultsNotFound **106**

Default configuration parameters have not been uploaded to the database. To correct this, you should set default values from the current values in the device using the *UploadConfigProperties* method of the *AppDevice* object. Use the lcaConfigPropOptSetDefaults and lcaConfigPropOptLoadUnknown upload options when you call the method. To avoid this error in other databases, import a program template with a XIF containing CP default values.

lcaErrNsCantFindConnection **107**

The requested connection was not found.

lcaErrNsLmobjMemNotFound **108**

The requested LonMark object member was not found. You will encounter this error if you try to use the *UnassignNetworkVariable* method to remove a network variable from a *LonMarkObject*, but the network variable was not previously assigned to the *LonMarkObject*. You can use the *AssignNetworkVariable* method to assign a network variable to a *LonMarkObject*.

lcaErrNsStaleFileHandle 109

An attempt was made to use a stale file transfer handle. This is an internal error, and should be reported to customer support.

lcaErrNsFileLimitReached 110

The file transfer handle limit has been reached.

lcaErrNsFileContention 111

A file could not be accessed due to contention with the initiator. You can avoid this error by not performing file transfers with devices that are already engaged in file transfers with other devices.

lcaErrNsAccessExpired 112

The LNS demonstration software has expired. A new copy must be obtained.

lcaErrNsCpValueNotFound 113

A value could not be found for the specified configuration parameter. This exception will be thrown if you use the Get*DataPoint* method to create a data point with the lcaDataSourceOptionsDatabaseOnly (2) option set, and then attempt to read the value of the data point, but the value does not exist in the OpenLNS database.

lcaErrNsNodeReset 114

An operation failure occurred due to an unexpected node reset.

lcaErrNsManagerNotAllowed 115

The specified services are not allowed by the manager. This is an internal error, and should be reported to customer support.

lcaErrNsUnsupportedService 116

Service attempted to reboot a node that is not a 3150.

lcaErrNsLabelNotFound 117

The requested label could not be found.

lcaErrNsDuplicateLabel 118

An object with this label already exists.

lcaErrNsNoRecoveryInProgress 119

There is no database recovery in progress for the recovery status service.

lcaErrNsCantModifyProgramIntf 120

The service attempted to modify a read-only program interface.

lcaErrNsCantModifyNvType 121

The service attempted to modify a read-only network variable type.

lcaErrNsNvNotLmobjMember **122**

The network variable is not a member of a LonMark object.

lcaErrNsHostResourceProblem **123**

There is a resource problem in the API or application.

lcaErrNsTxAlreadyCanceled **124**

The specified transaction was already canceled.

lcaErrNsFirmwareVersionMismatch **125**

The device contains an incompatible firmware version. This error will be generated if you attempt to load a new application image into a device with the Load method, and the system image (firmware version) used by the device is incompatible with the new application image. Some devices support the LoadEx method, which will upgrade the system image to a compatible version when this error occurs.

lcaErrNsUnimplementedCategory **126**

The service encountered an unimplemented category or category operation. This is an internal error, and should be reported to customer support.

lcaErrNsUnimplementedProperty **127**

The service encountered an unimplemented property or property operation. This is an internal error, and should be reported to customer support.

lcaErrNsDisallowedInMipMode **128**

The specified service or operation is disallowed because the interface is not an NSI. This is an internal error, and should be reported to customer support.

lcaErrNsDisallowedInTxHandler **129**

The specified service or operation is disallowed in the transaction notification handler. This is an internal error, and should be reported to customer support.

lcaErrNsNssNotInitialized **130**

The NsInit() function either was not called or failed. This is an internal error, and should be reported to customer support.

lcaErrNsNssEngineNotFound **131**

The NSS Win32 engine was not found. If you encounter this error, you should verify that LNS is installed properly on your machine.

lcaErrNsInvalidNsi **132**

The network interface is not a valid NSI mip.

lcaErrNsFileHeaderError **133**

The file which was accessed had an invalid header

lcaErrNsDbVersionError **134**

Returned when the NSS encounters a database (or non-volatile data) with the worng version number.

lcaErrNsNoNetworkInterface **135**

Returned when an attempt is made set the *MgmtMode* property to lcaMgmtModePropagateConfigUpdates (0) without selecting a network interface.

lcaErrNsNsiInuse **136**

Returned when an attempt is made to remove an NSI that is being used by other processes.

lcaErrNsDialingFailure **137**

Failed to call host. Probably due to busy, no answer, or recent failure.

lcaErrNsNetworkInterfaceState **138**

Local network interface is in the wrong state (e.g. unconfigured). When you encounter this error, set the network management mode to lcaMgmtModePropagateConfigUpdates (0) by writing to the *MgmtMode* property of the *System* object. Then, re-commission the network interface, if necessary.

lcaErrNsNetworkInterfaceConfig **139**

Local interface is not properly configured or updated. When you encounter this error, set the *MgmtMode* property of the *System* object to lcaMgmtModePropagateConfigUpdates (0). and recommission the network interface, if necessary.

lcaErrNsNetworkInterfaceInUse **140**

Network interface is currently in use.

lcaErrNsInvalidNeuronId **141**

The specified neuron ID is illegal. This error is reported if a Neuron ID containing zeroes in the middle 4 bytes is specified. Neuron Chips that use this format are defective and should be returned.

lcaErrNsDialupDataUninitialized **142**

Device not configured to dialup to host upon NSS communication.

lcaErrNsCannotStartAppl **143**

Dialup NSI was unable to start host application or NSS after connection.

lcaErrNsCapacityLimit **144**

License capacity or credit limit reached. This error will occur when all the credits in the license, including deficit credits, have been used. For more information on licensing, see Chapter 13 of the OpenLNS Programmer's Guide.

lcaErrNsCpNvLengthUnknown **145**

Length of configuration NV implementing a CP is unknown.

lcaErrNsInvalidErrorContext **146**

An update error had been reported prior to updating the nodes. This is an internal error and should be reported to customer support.

lcaErrNsLicenseViolation **147**

LNS license access failure. A variety of conditions may cause this error. Some relate directly to internal licensing components. For example, the license DLLs may not have installed properly, the files that identifies the license (nsseng.exe for the OpenLNS Server, lonmaker.DSL for LonMaker) do not exist, or the Crypkey licensing components (crypserv.exe and ckldrv.sys) are not running properly. You can resolve these problems by deleting the files that may be causing problems, and re-installing LNS. The license DLLs, as well as the crypserv.exe and ckldrv.sys files, can be found in the Windows "LNS Licenses" folder, and the Crypkey files can be found in the Windows "system32" folder.

Note that you must stop the "CrypkeyLicense" service and "NetworkX" driver before deleting the Crypkey files.

Certain conditions on the PC operating the license may also cause this error. For example, if the PC's file system is corrupted or low on space, or if the license files have been manually deleted, tampered with in any way, or moved by some disk-defragmentation utilities, this error will occur. You can resolve these situations by ordering a replacement license key. If the PC clock is set back to a time before the creation of the license or before the last time credits were purchased, this error will occur. In some cases, this can be resolved by rebooting the PC. Otherwise, it may be necessary to order a replacement key.

The error may occur if the license has been transferred out of the PC, manually terminated, or if there are zero maximum credits. In this case, you need to purchase additional credits for the license.

For more information on licensing, see Chapter 13 of the OpenLNS Programmer's Guide.

lcaErrNsLicenseExpired 148

LNS license time limit has expired. This error will occur if you are using a trial license and the number of days allocated to the license have expired, or if you are using a trial license and set the PC clock back.

For more information on licensing, see Chapter 13 of the OpenLNS Programmer's Guide.

lcaErrNsConflictWithCurrentNetwork 149

This error will be generated if you open a network that is already opened using a different network interface, NSS type, or database directory.

lcaErrNsConflictWithAnotherNetwork 150

This error will be generated if you open a network that specifies the same database directory as another currently opened network.

lcaErrNsBatchOptionNotImplemented 151

The batch service option combination specified is not implemented. This is an internal error, and should be reported to customer support.

lcaErrNsBatchNoResult 152

Access to specified batch result is not possible, since it does not exist. This is an internal error, and should be reported to customer support.

lcaErrNsLicenseFeatureDisabled 153

Feature has not been enabled in the license.

lcaErrNsDemoLicenseDisallowed 154

Full license already exists. Demo license not allowed.

lcaErrNsVnodeIndexMismatch 155

Virtual node index mismatch

lcaErrNsNiVniOpenFailure 156

Unable to open NI layer or VNI

lcaErrNsVniMsgError 157

VNI messaging error

lcaErrNsMaxDbOpened **158**

Exceeded the maximum number of databases

lcaErrNsEventsNotInitialized **159**

Event subsystem not initialized.

lcaErrNsMcsNotFound **160**

Monitor set cannot be found.

lcaErrNsMcsDblimit **161**

Max number of monitor points exceeded.

lcaErrNsMcpNotFound **162**

Monitor point cannot be found.

lcaErrNsMcpDblimit **163**

Max number of monitor points exceeded.

lcaErrNsNvmtInUse **164**

The network variable or monitor point is in use and cannot be used for current service.

lcaErrNsDbConversionInProgress **165**

The engine is in the process of converting a database. Internal

lcaErrNsIllegalMcpTarget **166**

Monitoring of specified target is not allowed.

lcaErrNsCantModifyNvName **167**

Can't modify network variable name.

lcaErrNsDbconvAccessFailure **168**

Unable to read/write old or new record during conversion

lcaErrNsUpgradeDisable **169**

Program interface upgrade has been disabled.

lcaErrNsUpgradeInfoNotFound **170**

No upgrade change info available for node.

lcaErrNsNssUpgradeRequired **171**

NSS program ID changed. Requires upgrade or revert back to old interface.

lcaErrNsEngineNotInitialized **172**

NSS/NSI engine initialization has not completed. This is an internal error, and should be reported to customer support.

lcaErrNsNiNoWinsockDll **173**

OpenLNS attempted to use IP without the winsock.dll file installed.

lcaErrNsNiCantOpenIpLink **174**

An OpenLNS network interface failed to open an IP connection.

lcaErrNsNiInvalidIpAddress **175**

An OpenLNS network interface uses an IP address not defined on this PC.

lcaErrNsNsiNotConfigured 176

NSI needs to be added or re-commissioned.

lcaErrNsNiDeviceOpenFailure 177

NI device driver open failure.

lcaErrNsBadLength 178

Invalid length of parameter, data, service, or message.

lcaErrNsBadName 179

Invalid name length, characters, or usage.

lcaErrNsBadDomain 180

Invalid domain length, index, or usage.

lcaErrNsInsufficientRtrsForMnc 181

No routers exist to complete logical path needed for monitoring and control.

lcaErrNsInsufficientRtrsForNsi 182

No routers exist to complete logical path needed for NSI and control. This exception will be thrown if you attempt to remove a router and the operation fails because it would break communication between the OpenLNS Server and a remote Full client.

lcaErrNs96BitAuthNotSupported 183

This error will be generated if you attempt to use 96-bit authentication on a device that does not support it.

lcaErrNsNeuronModelMismatch 184

Neuron model version mismatch. This error will be generated if you attempt to use the Load method to load an application image into a device that is incompatible with the device's system image. In this case, you need to upgrade the device's system image. Some devices support the LoadEx method, which automatically upgrades the system image before loading the application image if there are compatibility problems.

lcaErrNsSysimageFileFormatError 185

System image or upgrade image file format is invalid. Verify that OpenLNS has been installed correctly.

lcaErrNsSysimageCannotBeWritten 186

This error will be generated if you invoke the LoadEx method on a device, but the system image cannot be written, probably because it is not stored in flash memory. The old system image will remain intact if this error is thrown. However, the device will remain applicationless. You should load a new application image into the device with the Load or LoadEx methods in this case.

lcaErrNsSysimageUpgradeMemoryFailure 187

This error will be generated if there is a failure to write the new system image the device after the LoadEx method has been invoked. These failures usually occur because the device does not have flash memory in the required location to hold the new image during the download process. The old system image will remain intact if this error is thrown.

However, the device will remain applicationless. You should load a new application image into the device with the Load or LoadEx methods in this case.

lcaErrNsSysimageUpgradeFailed 188

This error will be generated when you invoke the LoadEx method on a device, and the new system image is successfully transferred to the device, but the switch from an old system image to a new system image fails. If you encounter this error, try invoking the LoadEx method again, or loading the previous system image back into the device.

s16BitUserErrorCode 189

The user defined error code is too big. This is an internal error, and should be reported to customer support.

lcaErrNs16BitWarningCode 190

The NSS warning code is too big. This is an internal error, and should be reported to customer support.

lcaErrNs16BitErrorCode 191

The NSS error code is too big. This is an internal error, and should be reported to customer support.

lcaErrNsFlexDomainAuthNotSupported 256

An attempt was made to leave a device authenticated but without a domain, and that device does not support flex domain authentication.

lcaErrNsCantModifyBoundNvType 257

Cannot modify the network variable type of a network variable when it is bound.

lcaErrNsCantModifyMonitoredNvType 258

Cannot modify network variable type of a network variable when it is being monitored.

lcaErrNsRegCannotOpenReg 259

Cannot open the NSS registry.

lcaErrNsRegUnknown 260

Unknown NSS registry error.

lcaErrNsRegConfigFileNotFound 261

The NSS registry backup file cannot be found.

lcaErrNsRegCannotUpdateConfigFile 262

Cannot update the NSS registry backup file.

lcaErrNsUnsupportedFileDirectoryVer 263

The device contains a version of the file directory that is unsupported.

lcaErrNsBadsiHdrSize 264

Invalid self-identification data header size.

lcaErrNsBadsiCapacityExceedsLimits 265

Capacity defined by self-identification data exceeds supported limits.

lcaErrNsBadsiCountExceedsCapacity 266

Resource count exceeds capacity limits in self-identification data.

lcaErrNsLmsdObjectRange 267

Invalid object range in the LonMark portion of a network variable self-documentation string. When specifying a range of objects, the lowest object index must appear before the highest object index.

lcaErrNsLmsdObjectMemberNumber 268

Invalid object member in the LonMark portion of a network variable self-documentation string. The member number must be between 1 and 32767.

lcaErrNsLmsdObjectArray 269

The LonMark portion of a network variable self-documentation string indicates that elements of the network variable are to be used as members of an object array. However, the network variable array has fewer members than the object array.

lcaErrNsLmsdVersion 270

Unsupported LonMark version number in the node self-documentation string.

lcaErrNsLmsdExpectedObjHdr 271

Expected LonMark object header in node self-documentation string.

lcaErrNsLmsdObjectNameTooLong 272

The LonMark Object name in the node self-documentation string exceeds 16 characters.

lcaErrNsCpsdOwnerType 273

Invalid CP owner type in header portion of the CP self-documentation string. The header must indicate whether the CP is owned by the device, one or more objects, or one or more network variables.

lcaErrNsCpsdMismatchedRangeTypes 274

Mismatched CP range types in the CP self-documentation string. Both the low and high range values must have the same type.

lcaErrNsCpsdDisableWithoutNodeobj 275

The CP Self-documentation of one or more CPs indicate that the object must be disabled in order to set the CP, but no node object is defined. Without a node object, it is not possible to disable an object.

lcaErrNsCpsdFileCpWithoutFtp 276

A CP template file is defined in the external interface file for a device, but the device does not support FTP or direct memory access.

lcaErrNsCpsdCpRange 277

The CP Self-documentation includes a range of objects or network variables, with the first index in the range being greater than the second. CP ranges must be specified from lowest to highest.

lcaErrNsCpsdDistributedArraySize 278

The CP Self-documentation indicates that a CP array should be distributed among range of network variables or objects, but the CP array size does not match the owner's range.

lcaErrNsCpsdTemplateVersion 279

The version of the CP template is unsupported.

lcaErrNsGroupDblimit 280

The number of LonTalk groups has been exceeded. There is a limit of 255 LONtalk groups per domain. In some case you may form connections using subnet or domain broadcast, to limit the use of groups. You may also limit the use of groups by using network variable aliases. These options are specified in the connection description template used when forming connections.

lcaErrNsNodeidDblimit 281

Number of LonTalk node IDs exceeded. There are 127 LONtalk node Ids per subnet. OpenLNS will allocate a new subnet, as needed, if the application does not specify the subnet to be used.

lcaErrNsNssEngineInitTimeout 282

The application timed out starting the NSS engine process (NssEng.Exe). This could be an indication that LNS is not installed properly. If you encounter this error, you should try rebooting your PC, or re-installing LNS.

lcaErrNsDmtNotFound 283

The dynamic message tag cannot be found.

lcaErrNsDmtDblimit 284

The maximum number of dynamic message tags has been exceeded. A single device can support up to 65,534 dynamic message tags, including internally defined message tags created for use with message points. You can add and remove message tags from a device with the Add and Remove methods of the *MessageTags* collection.

lcaErrNsLmobjNotDynamic 285

The specified *LonMarkObject* is not dynamic. This exception may be thrown if you attempt to assign a network variable to a static *LonMarkObject* with the *AssignNetworkVariable* method, unassign a network variable from a static *LonMarkObject* with the *UnassignNetworkVariable* method, or delete a static *LonMarkObject* with the Remove method.

You can determine if a *LonMarkObject* is dynamic or not by reading its IsDynamicproperty.

lcaErrNsLmobjNvNotDynamic 286

The specified network variable is not dynamic. This exception may be thrown if you attempt to assign a static network variable to a *LonMarkObject* with the *AssignNetworkVariable* method, unassign a static network variable from a *LonMarkObject* with the *UnassignNetworkVariable* method, or delete a static network variable with the Remove method.

You can determine if a network variable is dynamic or not by reading its *IsDynamic* property.

lcaErrNsLmobjInUse 287

The specified *LonMarkObject* is in use, and at least one of its member network variables is bound. This error may be thrown if you use the Remove or RemoveByIndex methods to delete a *LonMarkObject* with the lcaLonMarkObjectRemoveNVs (1) option set as the removalFlags element, and that *LonMarkObject* contains bound network variables.

The bound network variables must be disconnected before the *LonMarkObject* can be deleted. You can disconnect the network variables by specifying the lcaLonMarkObjectRemoveAndDisconnectNVs (3) option as the removalFlags element when you call the Remove or RemoveByIndex methods.

lcaErrNsApplReadWriteProtected 288

This exception will be thrown if you call the Load method to load a new application into a device, but the device's current application is read/write protected.

lcaErrNsEngineIsSuspended 289

The engine is currently suspended. This is an internal error, and should be reported to customer support.

lcaErrNsNotSupportedFromRemoteClient 290

The operation is not supported from a remote full client. This is an internal error, and should be reported to customer support.

lcaErrNsUnexpectedLink 291

The link between the parent and the child record was unexpected. This error usually indicates that the OpenLNS database has been corrupted. If you encounter this error, you should use the *Validate* method to run a database validation, and consider switching to a backup database.

lcaErrNsUnsupportedProgramCapability 292

The program defines capabilities that are not supported by OpenLNS. This error will be thrown when you call the Import method to import a device's XIF file, and the XIF file (or the device's self-documentation) indicates that it supports dynamic function blocks or dynamic function block members, but does not specify the EXTCAP_SUPPRESS_DYN_FB_DEF and EXTACP_SUPPRESS_DYN_FB_MBR_DEF flags. LNS 3.20 does not support downloading function blocks or function block membership to a device, any so any device that supports these operations needs to have an alternate way to define these dynamic function blocks and function block members. They must indicate that via the EXTCAP flags mentioned above.

lcaErrNsInvalidCpAttribute 293

The LONMARK organization does not allow modifiable device-specific configuration properties to be stored in files accessed via FTP on devices that support only sequential access. If you attempt to set the DeviceSpecificAttribute property to True on such a configuration property, this exception will be thrown.

lcaErrNsNotAllowedInASession 294

The operation is not allowed as part of a session. You can use sessions to group together properties and methods that must be performed together to avoid immediate error conditions. You can start a session with the *BeginSession* method, and close it with the *EndSession* method. You should note that the only network operations you can perform within sessions are those related to changes in the physical topology of your network. These operations include moving devices and routers, adding and removing routers, and setting the class of routers. For more information, see the online help for the *BeginSession* method.

lcaErrNsInvalidCpFileSize 295

The configuration property value file size reported by the device does not agree with the size defined by the definitions in the configuration property template file. Note that if

the device template was imported from an external interface file, the configuration property template file comes from the external interface file. Otherwise, the configuration property template file is loaded from the device.

lcaErrNsWarningFirst **4030**

A symbol denoting the beggining of the error number range reserved for warning codes.

lcaErrNsUpdateCommError **4030**

The database updates were successful, but one or more nodes were not updated because the NSS was unable to communicate with them. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsUpdateFuncError **4031**

The database updates were successful, but one or more nodes were not updated because the node(s) rejected the update, for example due to an authentication failure. This usually means that there is a configuration mismatch between the node and the NSS's database. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsNeuronStateChangeFail **4032**

The NSS's database updates were successful, but one or more nodes were not updated because the node(s) failed to make a necessary state transition. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsNodeUpdateDisallowed **4033**

The database updates were successful, but one or more nodes could not be updated because they are in the wrong state. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method. You can keep track of which devices are up to date using commissioning events and by reading the CommissionStatus property of each *AppDevice* object. If a node is applicationless, you must reload the application using the *AppDevice* .load method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving

persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsUpdateFileXferError 4034

The database updates were successful, but an update error occurred. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method. You can keep track of which devices are up to date using commissioning events and by reading the CommissionStatus property of each *AppDevice* object.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method. Most file update errors map to this status during an update.

lcaErrNsUpdateWhenUncnfg 4035

The database updates were successful, but one or more nodes could not be updated because they are not configured. You should ensure that all applicable devices are configured when this error occurs. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsUpdateWhenAppless 4036

The database updates were successful, but one or more nodes could not be updated because they are applicationless. You can load a node's application by calling the Load method on the *AppDevice* object associated with the node. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsUpdateCpFileTooShort 4037

The database updates were successful, but one or more devices was not updated because a configuration property file in the device is shorter than specified in the database. This is usually due to an inconsistency between the device's configuration property template file and its data file. The configuration property template file should be the same for all devices of a given type. If you encounter this type of error, check to make sure that the device is using the appropriate external interface file. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsUpdateCpFailure **4038**

The database updates were successful, but the configuration properties on one or more devices could not be updated as a result of invalid configuration property definitions. The NSS will continue to try to update the nodes in the background if the UpdateInterval property of the *System* object is set to a non-zero value, and you can force a retry with the RetryUpdates method.

You can keep track of which devices are up to date using commissioning events, and by reading the CommissionStatus property of each *AppDevice* object. If you are receiving persistent update failures on a device, you should re-commission the device with the *Commission* method.

lcaErrNsUpdatesDeferred **4039**

The database updates were successful, but the configuration properties on one or more devices could not be updated because the *MgmtMode* is currently set to lcaMgmtModeDeferConfigUpdates (1). You can update all the devices on the network with all pending updates by setting the *MgmtMode* property to lcaMgmtModePropagateConfigUpdates (0).

You can also update a single device with its pending device-only configuration changes while the *MgmtMode* property is still set to lcaMgmtModeDeferConfigUpdates (1) with the PropagateDeviceConfigUpdates method.

lcaErrNsWarningLast **4089**

A symbol denoting the end of the error number range reserved for warning codes.

Network Interface Errors

These errors are returned by the OpenLNS network interface. These errors will appear in the format NI: #<Error Number>. The LNS values for these errors are equal to the error number plus 16,500.

lcaErrNiNoDevice **1**

No network device is present.

lcaErrNiDriverNotOpen **2**

The network driver could not be opened.

lcaErrNiDriverNotInit **3**

The network driver required initialization

lcaErrNiDriverNotReset **4**

The network driver required a reset.

lcaErrNiDriverError **5**

An error occurred while communicating with the network driver.

lcaErrNiNoResponses **6**

No message was received during the wait time.

lcaErrNiResetFails **7**

The reset completion code was not received.

lcaErrNiTimeout **8**

Could not communicate with the network interface. A message timed out.

lcaErrNiUplinkCmd **9**

An uplink command was received from the NSS instead of a message.

lcaErrNiInternalErr **10**

An internal error occurred, probably an invalid completion code.

lcaErrNiFileOpenErr **11**

The log file could not be created or opened.

lcaErrNiNotNss **12**

The network interface received an LNS command, when no NSS was present.

lcaErrNiInvalidData **13**

The parameter or return data was invalid.

lcaErrNiMsgRouted **14**

The message was read, but then processed by a message callback function.

lcaErrNiResourceProblem **15**

The required resources were not available to complete the function.

lcaErrNiInvalidNsi **16**

The attached network interface does not contain NSI firmware.

lcaErrNiBadMessage **17**

Poorly formed message.

lcaErrNiVniSendMsgFailure **18**

The VNI had a message send failure.

lcaErrNiVniSendRespFailure **19**

The VNI failed to send a response.

lcaErrNiVniReadFailure **20**

The VNI failed to read a message.

lcaErrNiLostRefId **21**

The VNI lost a network interface reference message ID.

lcaErrNiVniNotOpen **22**

The LonTalk stack has not been opened.

lcaErrNiVniInitFailure **23**

Could not initialize the LonTalk stack.

lcaErrNiObsolete **24**

This is an internal error and should be reported to customer support.

lcaErrNiBadRefId **25**

Bad network interface reference message ID.

lcaErrNiImplicitAddrDisallowed **26**

May not send implicit tags via NI layer. Must use a message monitor point.

lcaErrNiUnimplemented **27**

This is an internal error and should be reported to customer support.

lcaErrNiBufferSize **28**

The message length is too long.

lcaErrNiNoMessages **29**

Message not found.

lcaErrNiNegativeResponse **30**

Negative response.

lcaErrNiStateError **31**

Unable to change the state of the network interface.

lcaErrNiReadBusy **32**

Previous response messages have not been read.

lcaErrNiMaxInterfacesOpen **33**

There are no network interface resources available.

lcaErrNiInvalidNiHandle **34**

Invalid network interface resource.

lcaErrNiOpenNiDeviceMismatch **35**

The network interface has already been opened with a different network driver. This error will be generated if you open a network that is already opened using a different network interface, NSS type, or database directory

lcaErrNiNoWinsockDll **36**

OpenLNS attempted to use IP without the winsock.dll file installed.

lcaErrNiCantOpenIpLink **37**

An OpenLNS network interface failed to open an IP connection.

lcaErrNiInvalidIpAddress **38**

An OpenLNS network interface uses an IP address not defined on this PC.

lcaErrNiDeviceOpenFailure **39**

Device driver open failure

lcaErrNiNoMoreLeft **40**

Signal iteration complete

lcaErrNiMipInUse **41**

If using a regular OpenLNS network interface (a non-high-performance network interface), only one network may be opened at a time. This error will be returned if you attempt to open a second network.

Connection Errors

Connection errors are errors returned when attempting to make network variable or message tag connections. These errors will appear in the format CONNERR: #<Error Number>. The LNS values for these errors are equal to the error number plus 16,500.

lcaErrConnInvalidTargetCount **1**

An incorrect number of targets were specified. You must specify at least one and no more than 25 targets when adding members. To add more members, you must invoke the *Connect* method multiple times.

lcaErrConnPropertyNotFound **2**

There is no *ConnectDescTemplate* object with the specified handle.

lcaErrConnConflictingProperties **3**

One or more of the fields of the connection descriptions used by intersecting connections are incompatible with one another. This may include any of the fields of a connection description, including the service type, the use of priority, and the use of authentication. Several of the fields of a connection description may cause additional exceptions to be thrown if those fields conflict. Those error codes start at CONNERR, #28 lcaErrConnConflictingPrioritySettings and end with CONNERR, #32 lcaErrConnConflictingBroadcastSettings. Note that these error codes (and the CONNERR, #3 lcaErrConnConflictingProperties exception) do not apply to monitor connections.

lcaErrConnNodeNotFound **4**

An invalid device handle was specified for the hub or for one of the targets.

Each device in the network has a unique handle that allows a client to identify the device. Once assigned, the handle is permanently associated with the device until that device is removed.

lcaErrConnNvmtOutOfRange **5**

The network variable or message tag index is larger than the maximum declared for that node.

The index is the number used to identify a message tag or a network variable. The Neuron C compiler assigns indices to message tags and network variables in the order in which they are declared. The first network variable declared in the code is assigned index 0, the second is assigned index 1, and so on. The same applies to the message tags.

lcaErrConnCantConnectNvToMt **6**

You can only connect network variables to network variables and message tags to message tags.

If you attempt to connect a message tag to a network variable, or vice versa, you will generate this error message.

lcaErrConnTypesDoNotMatch **7**

The network variables are not all of the same type.

If SNVTs are used for the connection then, all members must be of the same type.

lcaErrConnTypesDifferentSizes **8**

The network variables are not all of the same size.

If the connection members are not SNVTs, then all members must have the same length.

lcaErrConnDirectionOrPollingMismatch 9

An input network variable can only be connected to an output network variable, a polled output network variable can only be connected to a polling input network variable, and the implicit message tag msg_in can only be connected to an explicitly declared message tag.

This error will occur if an attempt was made to connect network variables of the same direction, network variables with mismatched polling attributes (a polled output network variable to non-polling input network variable), or msg_in to msg_in.

lcaErrConnMtagCantUseTurnaround 10

An attempt was made to connect message tags in the same node (i.e., to create a turnaround message tag connection, which is not possible).

lcaErrConnPriNotConfigurable 11

An attempt was made to modify the priority attribute; that attribute was not specified as configurable.

For network variables, the priority keyword may be included as part of the connection information in the Neuron C variable declaration. The config and nonconfig keywords specify whether the priority designation can be modified with a network management tool. The default is config.

The priority keyword affects output or polled input network variables.

Please refer to the Neuron C Reference Guide for more information.

lcaErrConnAuthNotConfigurable 12

An attempt was made to modify the authentication attribute and that attribute was not specified as configurable.

For network variables, the authentication keyword may be included as part of the connection information in the Neuron C variable declaration. Including the keyword config in the declaration allows the network management tool to change the authentication status of this network variable after a node has been installed. Setting nonconfig prevents the authentication status from ever being changed for this network variable. The default is config. Please refer to the Neuron C Reference Guide for more information.

lcaErrConnSvctypeNotConfigurable 13

An attempt was made to modify the protocol service type attribute, and that attribute was not specified as configurable.

For network variables, the service type can be specified in the connection information in the Neuron C variable declaration. The options are unackd, nackd_rpt and ackd. The keyword config is the default. This allows a network management tool to change the service specification at installation time. The keyword nonconfig indicates that the service type cannot be changed by a network management tool. Please refer to the Neuron C Reference Guide for more information.

lcaErrConnNvSelectorConstraint 14

At most, one input network variable in a device, and in some situations only one output network variable in a device, can use the same network variable selector, which is

shared by connections that intersect each other. This error will be generated when an attempt is made to form a connection that violates this constraint, and no network variable alias was available to overcome the constraint. If network variable aliases are available on the devices in the connection, they will normally be automatically used to avoid this error.

A selector is a number assigned to the network variable during the connection process in order to identify connected network variables. All members of the same connection, and any intersecting connections, share this selector. Without aliases, if an attempt is made to connect several inputs or several outputs (one or more of which polled by an input) on the same node in the same connection (or intersecting connections), this error will be generated. The use of aliases allows a single connection of several intersecting connections to have several different selectors (one per alias). If aliases are available, this error will normally not occur.

lcaErrConnNoMoreAddressSlots 15

This error will occur when a device involved in an attempted connection does not have an available address table slot to complete the connection. This can also occur when multiple connections are sharing an address table slot, and one of them changes. As a result, they can no longer share address table slots.

In some cases broadcast addressing can be used to reduce address table usage, because output connections to different nodes may be able to use the same broadcast address table entry. In addition, if the connection is a multicast connection, the use of broadcast addressing will not require a group address table entry on the target device. The use of broadcast messaging is controlled by the BroadcastOptions property of the *ConnectDescTemplate* object being used by the connection.

In some cases, aliases may be used to shift address table entry usage from one device to another. Multicast messages using a group use one address table entry on the source device, and one on each target device (note that this address table entry may be used for other connections as well). Using aliases to support multicast will result in using an address table entry for each target on the source node, but require no address table entries on the target node. The use of aliases for multicast is controlled by the AliasOptions property of the *ConnectDescTemplate* object.

lcaErrConnCannotCreateGroup 16

This error will occur when the requested connection requires a group, but there are no more groups available in the domain.

A domain can contain 256 groups. Groups and group membership are assigned when connections are created. If there are several targets within a connection, a group will normally be formed. The use of aliases may prevent a group from being used, and broadcast addressing can be used as an alternative to groups. Use of aliases for multicast and broadcast can be controlled via the AliasOptions and BroadcastOptions properties of the *ConnectDescTemplate* object being used by the connection.

Groups can be reused for multiple connections within the same domain. Connection timers used by the different connections must be compatible, and the collection of nodes must either be identical in all the connections, or all the connections must at least have the same set of destination nodes (the sources can be different). This last case is called group overloading. Reusing groups is done automatically with the OpenLNS Object Server.

lcaErrConnNoMoreNetvarSelectors 17

An attempted connection failed because all network variable selector values were in use.

A network can use up to 12288 selector values. Unrelated connections will normally require unique selector values. Intersecting connections automatically use the same selector *Network* variable aliases are used automatically to resolve selector conflicts on a node (i.e. to eliminate this exception from occurring), although this uses up one selector for each alias. Starting in LNS 2.0, selector values are grouped into categories, with one category being "shareable". Non-intersecting, point-to-point connections may reuse selector values from this category any number of times. This is done automatically by the OpenLNS Object Server.

lcaErrConnMtagInConnAlready 18

With the exception of the special message tag msg_in, a message tag may only appear in one connection. This error will occur when a connection was attempted that violated this constraint.

lcaErrConnAuthenticationMismatch 19

This error will occur when not all members of an attempted connection have the same setting for authentication.

It is possible to connect an authenticated output to a non-authenticated input, but an authenticated input must be connected to an authenticated output. A polling input must be authenticated if any of the targeted outputs are authenticated.

lcaErrConnGrpTooLargeForAckd 20

This error will occur when an attempt is made to form a connection of more than 64 members that uses acknowledged service. Only unacknowledged or unacknowledged/repeat service is allowed for connections with more than 64 members. Such a connection must be built using a connection description that specifies one of these service types.

This error may also occur when polled output network variables or polling input network variables are involved in the connection.

lcaErrConnUnackdWithAuth 21

An attempt was made to form a connection using authentication but not using the acknowledged service type.

The authentication process is composed of four messages, 2 of which are using the Acknowledged service. This is why using the authentication service without the Acknowledged service is illegal.

lcaErrConnAuthOnUnauthNode 22

An attempt was made to form an authenticated connection with an unauthenticated node. Only nodes with the *AuthenticationEnabled* set to True are permitted to participate in authenticated connections.

lcaErrConnBcastSubnetsDiffer 23

An attempt was made to subnet broadcast, but the devices are not all in the same subnet. This error is not currently generated, since the broadcast type used is determined automatically.

lcaErrConnNewBcastCausesNvLeak 24

An attempt was made to use broadcast addressing, but a network variable leak was detected on another node. A network variable leak means that update messages for the

network variable may be received by connections and devices that it is not intended to. LNS attempts to avoid this problem by the appropriate allocation of network variable selectors. However, some connection intersections make the use of broadcast addressing impossible, and so you may need to avoid using broadcast addressing in some situations. One solution for this problem is to use aliases for unicast connections, instead of using multicast connections. See the Optimizing Connection Resources section in Chapter 8 of the OpenLNS Programmer's Guides for more information on this.

lcaErrConnExistingBcastCausesNvLeak 25

An attempt was made to create a connection (of any type), but a detected network variable leak was caused by the use of broadcast addressing on another device. A network variable leak means that update messages for the network variable may be received by connections and devices that it is not intended to. LNS attempts to avoid this problem by the appropriate allocation of network variable selectors. However, some connection intersections are not possible when broadcast addressing is in use. One solution for this problem is to use aliases for unicast connections, instead of using multicast connections. See the Optimizing Connection Resources section in Chapter 8 of the OpenLNS Programmer's Guides for more information on this.

lcaErrConnBcastCantAck 26

Acknowledged services cannot be used with broadcast addressing.

Note that subnet broadcast is not allowed with ACKD service, and domain broadcast is not allowed with either ACKD or UNACKD_RPT services.

lcaErrConnDomainBcastMustUseUnackd 27

Domain broadcast addressing is restricted to the unacknowledged service.

Please note that UNACKD_RPT service is not allowed with Domain broadcast.

lcaErrConnConflictingPrioritySettings 28

The connection descriptions for intersecting connections have incompatible settings for priority. The priority option is available only for network variables. If an output network variable that is not declared as polled is a member of multiple connections, the priority settings for all of those connections must be identical. If a polling input network variable is a member of multiple connections, the priority settings for those connections must also be identical.

lcaErrConnConflictingAuthSettings 29

The authentication settings for the network variables in a connection are incompatible. If a target network variable (an input network variable, or polled output network variable) in a connection is using authenticated messaging, then all network variables involved in the connection must also use authenticated messaging.

lcaErrConnConflictingSvctypeSettings 30

The connection descriptions for intersecting connections have incompatible settings for service type. If an output network variable that is not declared as polled is a member of multiple connections, the service type settings for all of those connections must be identical.

lcaErrConnConflictingTimerSettings 31

The connection descriptions for intersecting connections have incompatible settings for one of the timer values. If an output network variable that is not declared as polled is a member of multiple connections, the timer settings for all of those connections must be

identical. If a polling input network variable is a member of multiple connections, the timer settings for those connections must also be identical.

lcaErrConnConflictingBroadcastSettings 32

The connection descriptions for intersecting connections have incompatible settings for broadcast addressing. If an output network variable that is not declared as polled is a member of multiple connections, the broadcast options for all of those connections must be identical. If a polling input network variable is a member of multiple connections, the broadcast options for those connections must also be identical.

lcaErrConnCantDirectlyManageMonitorConn 33

Monitor connections cannot be directly managed. This error indicates that the application is attempting to directly manage internal monitor connections, rather than implicitly manipulating those connections by using monitor points.

lcaErrConnConflictingMonitorProperties 34

The connection descriptions used for intersecting monitoring connections have conflicting settings. If the connection descriptions used by intersecting connections conflict, and neither connection is a monitoring connection, then any of the following errors may be thrown:

CONNERR, #3 lcaErrConnConflictingProperties

CONNERR, #28 lcaErrConnConflictingPrioritySettings

CONNERR, #29 lcaErrConnConflictingAuthSettings

CONNERR, #30 lcaErrConnConflictingSvctypeSettings

CONNERR, #31 lcaErrConnConflictingTimerSettings

CONNERR, #32 lcaErrConnConflictingBroadcastSettings

Object Server Errors

These errors are returned by the Object Server component. These errors will appear in the format LCA: #<Error Number>. The LNS values for these errors are equal to the error number plus 17,000.

lcaErrStringInvalidSize 1

A string passed did not match the required size, or exceeded the maximum size.

lcaErrInvalidType 2

An argument was passed with the wrong data type, or an assignment to a property has been made using an inappropriate data type.

lcaErrDuplicateKey 3

This error indicates that the object with this key or name already exists. Names or Keys of items being explicitly added to a collection object must be unique. It should be noted that this restrictions does not apply to implicitly added items.

lcaErrInvalidOleObject 4

An invalid OLE object has been passed to the object server by making assignments to LNS properties or calls to LNS methods.

lcaErrUninitializedDb **5**

Contact technical support.

lcaErrObjectNotFound **6**

An object has been requested by name, handle, or index, which could not be found in the databases. Typically, this would be a request from an OpenLNS collection object with a bad name, handle, or index.

It should be noted that this error can also occur in a few implicit assignments. For example, if the network interface to be used has not been explicitly assigned to the appropriate property, OpenLNS will continue using the network interface previously assigned to the *ActiveRemoteNi* or *NetworkInterface* property, respectively.

In such a scenario, this exception could occur even though no explicit assignment has been made, indicating that an implicit assignment maps to an unavailable object.

lcaErrCannotCreateObject **7**

The Object Server was unable to created the object.

lcaErrInternal **8**

An unexpected error occurred. Please contact technical support on LonSupport@Echelon.com

lcaErrFeatureNotAvailable **9**

A feature has been accessed which is not available in the current operating mode. Most likely, this error message is caused by an attempt to access properties which are not available to remote clients.

lcaErrFileOpen **10**

The Object Server is attempting to open a file that is already opened by another application.

lcaErrBadApbFile **11**

The file containing the binary version of the downloadable application image has been corrupted. The device manufacturer should be contacted for providing a valid APB file or the file should be re-built using the NEURON C compiler and linker.

Alternatively, the APB file can be re-build using the NXE32BIN.EXE utility, which itself requires the NXE and XFB files as input data.

lcaErrBadOrMissingXif **12**

The file containing the external interface information of the device has been is corrupted or is missing. The device manufacturer should be contacted for providing a valid XIF file, of said file should be re-build using using the NEURON C compiler and linker.

Alternatively, the device could be installed using an "Ad-Hoc" installation scenario, in which XIF files are not required. This requires the device to be available, online, functioning, and not having the self-documentation features disabled.

lcaErrBadOrMissingXfb **13**

The file containing the binary version of the external interface file has been corrupted or is missing. The device manufacturer should be contacted for providing a valid APB file, of said file should be re-build using using the NEURON C compiler and linker.

Alternatively, the XFB file might be re-build from the XIF file, using the XIF32BIN.EXE utility.

lcaErrInvalidPgmType 14

The *ProgramType* property contains an invalid value.

lcaErrInvalidCollectionIndex 15

A query to an OpenLNS collection object has been made using an invalid index or handle. Valid indices start with 1 and end with the current item count.

lcaErrCannotUpdateUnattachedObject 16

An attempt was made to update a device while not attached to the network.

lcaErrCannotUpdateWhenOffnet 17

An action has been requested which can not be performed in Offnet management mode. Change the *MgmtMode* property accordingly to allow this action to be performed.

lcaErrCannorRemoveReservedSubsystem 18

An attempt has been made to remove a reserved *Subsystem* object. Such *Subsystem* objects are all subsystem objects in the "ALL" and "Discovered" subsystem are and underneath.

Other read-only subsystems collection objects are the *Router* object's *Subsystems* property and the *AppDevice* object's *Subsystems* property.

lcaErrCannotAddToReservedSubsystem 19

An attempt has been made to add a subsystem object to a reserved subsystem area. Such areas are all subsystem objects in the "ALL" and "Discovered" subsystem are and underneath.

Other read-only subsystems collection objects are the *Router* object's *Subsystems* property and the *AppDevice* object's *Subsystems* property.

lcaErrInvalidPath 20

An invalid subsystem path has been specified.

lcaErrDatabaseExists 21

An attempt has been made to create a database which already exists. The database path should be changed to point to a different location.

lcaErrDatabaseNoAccess 22

Access to the database files has been denied by the operating system. Under operating systems with file security, the access permissions should be granted using the appropriate system tools.

lcaErrDatabasePathTooLong 23

The database path passed to the *Networks* collection's *Add* method is too long. The path to a network database must not exceed 26 characters.

lcaErrMaxClientsExceeded 24

The LNS maximum of 254 simultaneous clients has been exceeded.

lcaErrDatabase **25**

An unexpected database error occurred. Please take a note of the full error message, and contact technical support at LonSupport@Echelon.com

lcaErrInvalidDeviceTemplate **26**

The *DeviceTemplate* object has been corrupted.

lcaErrNoDatabase **27**

The network database has not been found in the location previously registered.

lcaErrNoDbDictionary **28**

The LNS dictionary files _objects.dat and _objects.idx are missing or can not be found. The default location is C:\LONWORKS*ObjectServer*\Dictionary.

lcaErrInvalidSystemObject **29**

The *System* object associated with this network database is invalid, the network database must be rebuilt from scratch.

lcaErrNetworkDbNotOpen **30**

A service has been requested prior to invoking the *Network* object's *Open* method.

lcaErrCannotRemoveSubsystemWithDevices 31

An attempt to remove a subsystem which still contains *AppDevice* or *Router* objects has been made. These objects must be removed first.

lcaErrObjectLocked **32**

An attempt has been made to modify a locked object. Objects can be locked and unlocked with the *Lock* method.

lcaErrCantCreateDbServer **33**

This error is returned by a remote client when the host is unable to launch the OpenLNS Server.

lcaErrUnsupportedEvent **34**

Contact customer support.

lcaErrGeneric **35**

Contact customer support.

lcaErrDbServer **36**

An OpenLNS Server error. Contact customer support.

lcaErrCommissionCausesReplace **37**

An attempt has been made to commission a device after changing it's Neuron ID. In this situation, the *Replace* method should be used.

lcaErrAppdeviceRequired **38**

The Object Server was unable to communicate with the Appdevice.

lcaErrWrongWriteDataSize **39**

Contact customer support.

lcaErrInvalidFormat **40**

Contact customer support.

lcaErrCompiler 41

Generic compiler error. See the LNS Field Compiler documentation for details.

lcaErrProjectMgrNotAvailable 42

The LNS Field Compiler's project manager and dependency checker is unavailable. See the LNS Field Compiler documentation for details.

lcaErrInvalidLinkerOptions 43

The options passed to the NEURON Linker were invalid. See the LNS Field Compiler documentation for details.

lcaErrInvalidBuildTemplate 44

The *BuildTemplate* object is invalid, as it does not comply with the requirements and constrains to the LNS Field Compiler. See the LNS Field Compiler documentation for details.

lcaErrInvalidHardwareTemplate 45

The *HardwareTemplate* object is invalid, as it does not comply with the requirements and constrains to the LNS Field Compiler. See the LNS Field Compiler documentation for details.

lcaErrInvalidProgramTemplate 46

The *ProgramTemplate* object is invalid, as it does not comply with the requirements and constrains to the LNS Field Compiler. See the LNS Field Compiler documentation for details.

lcaErrInvalidExportOption 47

The options passed to the NEURON Exporter were invalid. See the LNS Field Compiler documentation for details.

lcaErrCantCallDtssComponent 48

An attempt to communicate with a LNS Device Control failed.

lcaErrCantLoadDevice 49

The application could not be loaded into the device.

lcaErrCantRemoveFromReservedSubsystem 50

An attempt has been made to remove an item from a reserved subsystem. Such subsystem objects are all subsystem objects in the "ALL" and "Discovered" subsystem are and underneath.

Other read-only subsystems collection objects are the *Router* object's *Subsystems* property and the *AppDevice* object's *Subsystems* property.

lcaErrDeviceTemplatePgmMismatch 51

The program ID from the *DeviceTemplate* object does not match the one found on the device during an attempt to commission the device.

lcaErrMustSetNeuronIdFirst 52

The NEURON ID must be set prior to querying this property or invoking the method.

lcaErrInvalidOnDiscoveredDevice 53

lcaErrCantCopyNssDirectory **54**

Internal error. Contact customer support.

lcaErrCantRecoverNotNewDb **55**

A call to the *RecoverFromNetwork* method has been made, using a previously used network database. For the recovery process, a newly created network database is required.

lcaErrCantRecoverMissingNssDbFiles **56**

Some parts of the network database are lost and can not be recovered. The network must be re-build from scratch, should no backup be available.

lcaErrCompilerInit **57**

Initialization of the NEURON C Compiler failed. See the LNS Field Compiler documentation for details.

lcaErrAssembler **58**

Generic error message, indicating an unspecified NEURON Assembler error. See the LNS Field Compiler documentation for details.

lcaErrAssemblerInit **59**

Initialization of the NEURON Assembler failed. See the LNS Field Compiler documentation for details.

lcaErrLinker **60**

Generic error message, indicating an unspecified NEURON Linker error. See the LNS Field Compiler documentation for details.

lcaErrLinkerInit **61**

Initialization of the NEURON Linker failed. See the LNS Field Compiler documentation for details.

lcaErrExport **62**

Generic error message, indicating an unspecified NEURON Exporter error. See the LNS Field Compiler documentation for details.

lcaErrExportInit **63**

Initialization of the NEURON Exporter failed. See the LNS Field Compiler documentation for details.

lcaErrGlobalDbNotOpen **64**

An attempt has been made to obtain a service from LNS prior to opening the LNS global database. This database is opened and closed by opening and closing the *ObjectServer* object itself.

lcaErrOnlyAllowedOnHostIntf **65**

The requested operation is only allowed on *Interface* objects associated with host devices. Typically, this exception indicates an attempt to create virtual interface objects or to dynamically add or remove network variables from regular, non-hosted, LONWORKS devices.

lcaErrAnotherNetworkAlreadyOpen 66

An attempt to open a network database failed because another network object is still held open by the same client process. This error should not occur in LNS 3 or greater. If it does, contact customer support.

lcaErrSystemNotOpen 67

A service has been accessed prior to successfully opening the system object, which requires the system to be open.

lcaErrDbConversionRequired 68

The global database or local database is from an older version of LNS, and the appropriate *CompactDb* method must be called prior to opening the object. The CompactDb method will compact the database, reorganize its contents, and automatically upgrade to the latest version.

lcaErrPropertyCantBeSetWhenOpen 69

An attempt has been made to change a property while the associated key object (the *System* , *Network*, or *ObjectServer* object) is open. This property requires the object to be closed prior to accepting changes.

lcaErrCantRecoverReadOnlyFile 70

Contact customer support.

lcaErrStringIsEmpty 71

Contact customer support.

lcaErrWrongServerDll 72

Contact customer support.

lcaErrInvalidProgramType 73

Contact customer support.

lcaErrNotAvailableOnRemoteClient 74

The requested property of method is not available on the remote client.

lcaErrDuplicateApplicationName 75

The value assigned to the *ApplicationName* property was invalid.

lcaErrCantRecoverNotAttachedToNetIntf 76

Recovery fails due to a lack of an OpenLNS network interface.

lcaErrDevTemplateWithThisProgramExists 77

Implicit or explicit creation of a *DeviceTemplate* object failed due to the program ID already being in use with a different device template in the same network database.

The program ID is a unique identifier for each device type (i.e. device template object). Devices certified by the LonMark Interoperability Association will have a unique program id, using a globally unique manufacturer ID allocated by LonMark.

lcaErrCantCreateLcaServer 78

The OpenLNS Server cannot be created.

lcaErrCantLaunchLcaServer **79**

The OpenLNS Server cannot be launched.

lcaErrNoMemory **80**

No more system memory available.

lcaErrDbInvalidVersion **81**

The database has an invalid version. Note that OpenLNS databases can be upgraded using the *CompactDb* method, but also note that OpenLNS databases are not backwards compatible. Once updated to version N+1, the database will not be accessible for a version N client any more.

lcaErrDictionaryInvalidVersion **82**

The OpenLNS database dictionary files have an invalid version.

lcaErrDbAndDictionaryMismatch **83**

The LNS dictionary files do not match the requirements of the global database and network database. After making a backup copy, try upgrading the databases using the *CompactDb* method. Note that OpenLNS databases can be upgraded using the CompactDb method, but also note that OpenLNS databases are not backwards compatible. Once updated to version N+1, the database will not be accessible for a version N client any more.

lcaErrCouldNotOpenNeuronCFile **84**

The NEURON C source code file could not be opened. It may be locked by another application, it may be corrupted, or the operating system's security mechanism might prevent LNS from accessing the file.

lcaErrCouldNotOpenLangResourceFile **85**

The language resource file, which is part of the LonMark device resource files, could not be opened. It may be locked by another application, it may be corrupted, or the operating system's security mechanism might prevent LNS from accessing the file.

lcaErrCouldNotOpenFuncProfileFile **86**

The functional profile template file, which is part of the LonMark device resource files, could not be opened. It may be locked by another application, it may be corrupted, or the operating system's security mechanism might prevent LNS from accessing the file.

lcaErrStringInvalidChar **87**

The string passed contains one of more characters which are invalid in this environment. For name properties, the file name style conventions from the windows operating system apply.

lcaErrAuthKeyNotSet **88**

The authentication key has not been set prior to accessing authenticated objects. Such an object can be any *AppDevice* or *Router* object, including the OpenLNS Server itself. It should be noted that a remote client needs to specify the authentication key prior to opening the (remote) system.

lcaErrReadOnlyCollection **89**

An attempt to change a member of a write-protected collection has been made. Such read-only collection objects are the ones which by design do not support adding or

removing items at runtime. It should be noted that this exception may also occur with collections which are usually write-enabled, as the context might not allow for changes.

For example, the *Subsystems* collection would typically be seen in the hierarchy *System* ::*Subsystems* context, to which further items can be added using the *Subsystems* collection's *Add* method. The *AppDevice* and Router object's *Subsystems* collection objects, however, represent all subsystem objects to which the device belongs to. Such a collection would be a read-only collection in this particular context, and a write-enabled collection in another context.

lcaErrValueOutOfRange **90**

The value given exceeds a range limit.

lcaErrCantUpdateWhenPaused **91**

A requested service can not be completed due to the DataServer being paused. *DsPause* should be set to FALSE to re-start the DataServer.

lcaErrCantCreateLcaMonitor **92**

The Lca Monitor, a utility component for internal purposes, could not be created. This failure is caused by a lack of memory available to LNS.

lcaErrCantLaunchLcaMonitor **93**

The Lca Monitor, a utility component for internal purposes, could not be launched. This utility, LCAMON.EXE, would typically be located in the C:\LONWORKS\Bin folder, which must be part of the search path or otherwise visible from the working directory of the calling tool. The 93 exception would indicate any problem launching the LCA Monitor utility tool like memory shortage, bad permissions, executable not found, etc.

lcaErrMustSetDsPauseToChange **94**

The DataServer must be paused prior to applying the requested changes. Use *DsPause* to pause the DataServer.

lcaErrNotAvailableOnLocalClient **95**

A service only available to remote clients has been requested from a local client.

lcaErrObjectDeleted **96**

The object has been deleted.

lcaInvalidRmcObject **97**

Invalid object received through remote procedure call. Contact customer support.

lcaInvalidRmcServer **98**

Contact customer support.

lcaInvalidRmcClient **99**

Contact customer support.

lcaInvalidRmcVersion **100**

The client and server are running different versions of LNS and the Object Server was unable to compensate.

lcaInvalidRmcMethod **101**

Contact customer support.

lcaNeuronIdInUse **102**

The NEURON ID assigned is already in use for another device in the network.

lcaDbCantBeCompactedWhenOpen **103**

CompactDb() methods must be called prior to opening the associated object.

lcaDbCantBeRemovedWhenInUse **104**

The object can not be removed as it still is in use.

lcaErrCantRemoveDefaultAccount **105**

The default *Account* object can not be removed.

lcaErrCantRemoveAccountWithCharges **106**

An account object needs to be empty for being removed.

lcaErrCantRemoveNssNode **107**

The device can not be removed, as it is associated with the network interface currently in use

lcaErrCantConnectToServer **108**

The remote client fails to connect to the OpenLNS Server

lcaErrTcpSocketError **109**

An unspecified socket error occurred, causing the IP connection to get lost

lcaErrBadPermissionFormat **110**

Malformed data was assigned to the *PermissionString* property.

lcaErrCantModifyInSharedMode **111**

An attempt has been made to modify a shared DataServer property. Change the *DsMode* property to enter exclusive mode, or refrain from modifying the property.

lcaErrDataUnderrun **112**

An IP remote client did not receive expected data in time. The connection might be lost.

lcaErrDataOverrun **113**

An IP remote client received unexpected data. The configuration should be verified.

lcaErrIllegalInTransaction **114**

A service, which can not be part of an explicit transaction, has been requested within an explicit transaction. Complete or dismiss the transaction using the *CancelTransaction* method or *CommitTransaction* method prior to invoking this service.

lcaErrNoPermission **115**

Permission to an item has been requested but denied. Use the *PermissionString* property local to the server to change permission preferences.

lcaErrStaleObject **116**

An attempt was made to modify or use a stale OpenLNS Object.

For example, once the system object has been successfully closed, all objects accessed through the system object will be stale, and should be de-referenced.

lcaErrNotAvailableOnTcpClient **117**

A service which is only available local to the server or on a Full client has been requested from a Lightweight client.

lcaNotAnNsiHost **118**

The (hosted) device does not execute the NSI firmware, and the requested service can not be completed therefore.

lcaErrInterfaceNotModifyable **119**

An attempt to dynamically modify a static interface was made. To support dynamic interface modifications, dynamic network variables might be supported by implementing the wink command extensions as documented in the LonMark Application Layer (7) Interoperability Guidelines, Version 3.1 or better.

lcaErrNoConnectionToServer **120**

There is no communication with the OpenLNS Server. This error may occur for a remote client operation when the OpenLNS Server has shutdown.

lcaErrInvalidInitString **121**

The *AddNvFromString* or *InitFromString* method was provided with an invalid string.

lcaErrReadOnlyInContext **122**

Contact customer support.

lcaErrInvalidPropChangeEvent **123**

Contact customer support.

LcaNotAllowedWhenIndependent **124**

The called service doesn't work in server-independent mode.

LcaNetworkNotOpen **125**

The network is not open.

LcaCantReadMsgPoint **126**

The object server is unable to read the message point.

LcaOpenCloseTypeMismatch **127**

If the *Network* object's *Open* method is used to open a network, the *Close* method must be used to close it. If the *OpenIndependent* method is used to open the network, the *CloseIndependent* method must be used to close it.

LcaInvalidDomainLength **128**

The domain is the wrong length.

LcaDatabaseRecoveryFailed **129**

The database recovery operation failed.

LcaOnlyAllowedOnLocalVNI **130**

The requested service is not available on a remote Full client.

LcaInvalidHexValue **131**

An invalid hexadecimal value was supplied.

LcaUniqueNVNameRequired **132**

A network variable name is required for this service.

LcaInternalErrBadExceptionCode **133**

Internal error.

LcaInternalErrBadHresultExceptionCode **134**

Internal error.

LcaInternalErrBadVNIDataServer **135**

Internal error.

LcaInternalErrBadNsdHandle **136**

Internal error.

LcaInternalErrStartTransaction **137**

Internal error.

LcaNotAllowedOnVniNetwork **138**

An attempt was made to call the *OpenIndependent* method on a *Network* object which was not fetched from the *VNINetworks* collection.

LcaInvalidAddressingMode **139**

When using the *GetMessagePoint* method, an invalid addressing mode was supplied.

LcaNotAllowedOnPermanentNsd **140**

An attempt was made to remove a permanent network service device.

LcaOnlyAllowedWhenIndependent **141**

The requested service can only be performed on networks opened with the *OpenIndependent* method.

LcaDatabaseDiskFullErr **142**

The disk containing the OpenLNS database is full.

lcaErrUninitialized **168**

There is uninitialized data.

lcaErrInvalidHandle **169**

An object has an invalid handle.

lcaErrInvalidProgramId **170**

An object has an invalid program ID.

lcaErrOutOfHandles

OpenLNS ran out of handles. **171**

lcaErrNotImplemented

The error is not implemented in this version of OpenLNS **172**

lcaErrCantRemoveExtensionWithChildren **173**

Cannot remove an extension when it has child extensions. Remove the child extensions first.

lcaErrExtensionTooLargeForRemote **174**

The extension record is too large to be accessed remotely.

Data Server Errors

These are errors returned by the LNS Data Server. The Data Server handles network variable and message tag monitoring and control. These errors will appear in the format DS: #<Error Number>. The LNS values for these errors are equal to the error number plus 18,000.

lcaErrDsBusy **1**

Could not get semaphore to perform Data Server operation.

lcaErrDsAlreadyStarted **2**

Cannot start Data Server, as it was already started.

lcaErrDsNoServer **3**

Invalid Data server handle passed in.

lcaErrDsOptionOutOfRange **4**

Invalid option passed in.

lcaErrDsNotImplemented **5**

Function not implemented.

lcaErrDsNotStarted **6**

Function unavailable until the Data Server has been started.

lcaErrDsCantActivate **7**

Cannot activate

lcaErrDsCantDeactivate **8**

Cannot deactivate.

lcaErrDsIllegalObjectHandle **9**

Illegal object handle was passed in.

lcaErrDsReentry **10**

Tried to perform an exclusive operation, such as updating a particular network variable on a node, that was already in progress.

lcaErrDsRegistrySetErr **11**

Unused.

lcaErrDsAlreadyCreated **12**

Tried to create a Data Server when one already existed for this process.

lcaErrDsException **13**

A system exception was thrown.

lcaErrDsInvalidObjectType **14**

Unused.

lcaErrDsUserCancel **15**

The operation specified was canceled interactively at the user interface.

lcaErrDsVersionIncompatibility 16

Some of the DLLs used by the Data Server are older, incompatible versions.

lcaErrDsInitValidationError 17

This indicates that the Data Server failed to initialize properly. This exception could be thrown if your application specifies an invalid name or index number when retrieving an object from the OpenLNS database.

lcaErrDsUpdateRegistryFailure 18

Could not update the Windows registry.

lcaErrDsNsCreateClientFailed 19

Failed to create a *Network* Services client for data services.

lcaErrDsNsInitFailed 20

Failed to initialize the *Network* Services Server.

lcaErrDsNsNotOpen 21

Error on shutdown, when attempting to close the *Network* Services server, the server is not open.

lcaErrDsGetHandleFromNameFail 22

Could not get the node handle from its name.

lcaErrDsInvokeNodeAddrFail 23

Could not get the node's address on the network from the *Network* Services server.

lcaErrDsGetRegistryFailure 24

Failed to retrieve an item from the Windows registry.

lcaErrDsSetRegistryFailure 25

Failed to write an item to the Windows registry.

lcaErrDsInvalidRegKey 26

Failed to set a Windows registry key because the new key passed in was invalid.

lcaErrDsInvalidNodeName 27

An invalid node name was specified, or no name services provider was specified for the NSS.

lcaErrDsNodeDoesNotExist 28

Node does not exist on this network, probably because it has not yet been configured with a network address.

lcaErrDsNvDoesNotExist 29

A network variable operation was attempted on an NV that does not exist.

lcaErrDsNvGetConnInfoFailed 30

A *Network* Services call to get connection information for this network variable has failed.

lcaErrDsNvGetTypeInfoFailed 31

A *Network* Services call to get type information for this network variable has failed.

lcaErrDsStartNvObjectFailed 32

Could not start the network variable object.

lcaErrDsDeleteNvObjectFailed 33

Could not delete the network variable object.

lcaErrDsNvSetObjDataFailed 34

Failed to write data to the network variable object.

lcaErrDsDataFormatFailure 35

An attempt to format the requested data has failed.

lcaErrDsCreateThreadFailure 36

Could not create the thread to receive LONWORKS messages from the network.

lcaErrDsCreateEventFailure 37

Could not create the event to signal arriving LONWORKS messages from the network.

lcaErrDsEventSignalTimeout 38

Timed out waiting for a synchronizing event to be signaled by a LONWORKS network response.

lcaErrDsCannotForceDataRequest 39

Unused.

lcaErrDsNvIllegalRetry 40

The number of retries specified for this network variable is out of range.

lcaErrDsNvIllegalInterval 41

The interval specified for this network variable is illegal.

lcaErrDsInvalidDriverName 42

The network interface driver specified does not exist.

lcaErrDsMismatchTypeSize 43

The type specified is not the correct size to represent this network variable.

lcaErrDsFmtTypeNotFound 44

The format type referenced does not exist.

lcaErrDsNvIllegalField 45

The network variable field referenced does not exist.

lcaErrDsInvalidServerHandle 46

The Data Server handle passed in is invalid.

lcaErrDsOptionIsReadOnly 47

Option is read-only. The value may not be set.

lcaErrDsObjectNotStarted 48

Failed because the object referenced must be started first.

lcaErrDsObjectIsAlreadyStarted **49**

Cannot start this object, as it is already started.

lcaErrDsSetInvalidServiceType **50**

Tried to set the service type to an invalid value.

lcaErrDsNiIsBusy **51**

Operation failed because the network interface was busy.

lcaErrDsNiOpenFailure **52**

Error opening the network interface.

lcaErrDsNiUnknown **53**

Error with the network interface.

lcaErrDsCantWriteRemoteNvOutput **54**

It is not possible to write a value to a remote output network variable.

lcaErrDsFormattingInternalError **55**

Error loading one of the SNVT or User type (.TYP) files.

lcaErrDsFmtFileLoadFailure **56**

Error loading one of the SNVT or User format (.FMT) files.

lcaErrDsBufferTooSmall **57**

The specified buffer is too small for the current operation.

lcaErrDsMsgSendFailure **58**

Failed to send the current message.

lcaErrDsNoDataAvailable **59**

No data is available for retrieval in the current operation.

lcaErrDsTypeFileNotFound **60**

The specifed type file could not be found.

lcaErrDsOptionIsMsgobjOnly **61**

The specified configuration option only applies to message objects.

lcaErrDsOptionIsNvOnly **62**

The specified configuration option only applies to network variable objects.

lcaErrDsNoResource **63**

System resources unavailable.

lcaErrDsInvalidParamValue **64**

One or more of the passed parameters has an illegal value.

lcaErrDsMtagDoesNotExist **65**

The specified message does not exist.

lcaErrDsRegistryDataInvalid **66**

Invalid data was found in a Data Server registry entry.

lcaErrDsFmttypeHasNoFields **67**

The format type specified does not contain information about fields.

lcaErrDsEventSubscriptionFailed **68**

Failed to subscribe to the specified event.

lcaErrDsCantFindFormatCatalog **69**

The format catalog could not be found.

lcaErrDsCantFindStandardTypeFile **70**

The specified standard type file could not be found.

lcaErrDsEventDesubscriptionFailed **71**

Failed to unsubscribe to the specified event.

lcaErrDsInvalidVersion2TypeName **72**

An invalid Version2 format name was supplied. Version 2 type names must start with SNVT, UNVT, SCPT or UCPT. Format names should be taken from the standard and user-defined resource file catalogs located in the directory specified by the LdrfCatlalogPath property.

lcaErrDsInvalidWhenNvFieldInUse **75**

The requested operation must be performed on a network variable. It cannot be performed on an individual network variable field.

lcaErrDsInvalidWhenNvFieldNotSpecified **76**

The requested operation must be performed on an individual network variable field. It cannot be performed on a network variable.

lcaErrDsGetAppmanagerFailed **77**

Failed to specify the application manager.

lcaErrDsCantSetHostMsgOwner **78**

Fatal error occurred when attempting to become host message owner. Only one process can be host message owner at any time.

lcaErrDsFmtIllegalBaseType **79**

Format specifier is not valid for the type/field.

lcaErrDsNetworkError **200**

Failed to communicate with target device.

lcaErrDsTargetNodeFailure **201**

Target node responded with failure code.

lcaErrLnsDsUnexpected **365**

Unexpected failure. Contact customer support.

lcaErrLnsDsFailed **366**

Generic failure. Contact customer support.

lcaErrLnsDsOutOfMemory **367**

Memory allocation failure.

lcaErrLnsDsNotImpl **368**

Not implemented.

lcaErrLnsDsPointer **369**

Invalid pointer.

lcaErrLnsDsInvalidArg **370**

Invalid argument. One way this error can occur is if you call the *Write* method on a *DataPoint* object obtained from a *NetworkVariable* or *NvMonitorPoint* object before reading or setting its value with the *DsIsDefaultFormat* property, the *RawValue* property, or the *Value* property. This indicates that the *DataPoint* 's buffer is empty.

lcaErrLnsDsRange **371**

Argument supplied is out of valid range.

lcaErrLnsDsTimeout **372**

Timeout occurred.

lcaErrLnsDsSizeTooSmall **373**

Size is too small.

lcaErrLnsDsInvalidObject **374**

Object not valid.

lcaErrLnsDsInvalidWindow **375**

Invalid window.

lcaErrLnsDsInvalidOperation **376**

Operation not valid, or the operation was attempted on an invalid object.

lcaErrLnsDsQueueFull **377**

The message queue is full and the most recent message has been dropped.

lcaErrLnsDsQueueStopped **378**

The message queue was stopped by a client application and the most recent message has been dropped.

lcaErrLnsDsOutstandingRefs **379**

Outstanding object references still exist.

lcaErrLnsDsOutstandingClients **380**

Outstanding clients still exist.

lcaErrLnsDsOutstandingNetworks **381**

Outstanding networks still exist.

lcaErrLnsDsOutstandingVNIs **382**

Outstanding VNIs still exist.

lcaErrLnsDsOutstandingMonitorSets **383**

Outstanding monitor sets still exist.

lcaErrLnsDsNotFound **384**

Object not found.

lcaErrLnsDsClientNotFound **385**

Client not found.

lcaErrLnsDsNetworkNotFound **386**

Network not found.

lcaErrLnsDsVniNotFound **387**

VNI not found.

lcaErrLnsDsMonitorSetNotFound **388**

Monitor set not found.

lcaErrLnsDsMonitorPointNotFound **389**

Monitor point not found.

lcaErrLnsDsNodeNotFound **390**

Node not found.

lcaErrLnsDsNetworkVariableNotFound **391**

Network variable not found.

lcaErrLnsDsAppMessageNotFound **392**

Application message not found.

lcaErrLnsDsNotOpen **393**

Object must be opened before the requested operation can be performed.

lcaErrLnsDsNetworkNotOpen **394**

Network must be opened before the requested operation can be performed.

lcaErrLnsDsVniNotOpen **395**

VNI must be opened before the requested operation can be performed.

lcaErrLnsDsMonitorSetNotOpen **396**

Monitor set must be opened before the requested operation can be performed.

lcaErrLnsDsNotConnected **397**

The client must be connected to the server before the requested operation can be performed.

lcaErrLnsDsOpenFailed **398**

There was an error opening the object.

lcaErrLnsDsVniOpenFailed **399**

There was an error opening the VNI.

lcaErrLnsDsMonitorSetOpenFailed **400**

There was an error opening the monitor set.

lcaErrLnsDsCloseFailed **401**

There was an error closing the object.

lcaErrLnsDsVniCloseFailed **402**

There was an error closing the VNI.

lcaErrLnsDsThreadCreateFailed **403**

There was an error creating the thread.

lcaErrLnsDsNotTemporaryMonitorSet **404**

The requested operation can only be performed on temporary monitor sets.

lcaErrLnsDsNotTemporaryMonitorPoint **405**

The requested operation can only be performed on temporary monitor points.

lcaErrLnsDsNssNotOpen **406**

NSS must be opened before the requested operation can be carried out.

lcaErrLnsDsNssCreateClientFailed **407**

There was an error creating the NSS client.

lcaErrLnsDsNssInitFailed **408**

There was an error initializing NSS.

lcaErrLnsDsNssException **409**

Error occurred during an NSS operation.

lcaErrLnsDsReadFailed **410**

Network variable read failed.

lcaErrLnsDsWriteFailed **411**

Network variable write failed.

lcaErrLnsDsPollFailed **412**

Network variable poll failed.

lcaErrLnsDsSendFailed **413**

Application message transmission failed.

lcaErrLnsDsMsgRejected **414**

Application message rejected, as network management commands are restricted.

lcaErrLnsDsRequestFailed **415**

Application message request failed.

lcaErrLnsDsResponseFailed **416**

Application message response failed.

lcaErrLnsDsAddressNotAvailable **417**

The source node address is not available.

lcaErrLnsDsIllegalHexCharacter **418**

An illegal hexadecimal character was specified in the string.

lcaErrLnsDsTooManyClients **419**

There are too many clients connected to the server.

lcaErrLnsDsInvalidClientContext **420**

Invalid or no client context was supplied.

lcaErrLnsDsImplicitAddress **421**

The requested operation is not allowed, as the monitor point uses implicit addressing.

lcaErrLnsDsExplicitAddress **422**

The requested operation is not allowed, as the monitor point uses explicit addressing.

lcaErrLnsDsNotOwner **423**

The requested operation is not allowed, as the client is not the owner of the object or process affected.

lcaErrLnsDsNoNameServicesManager **424**

Cannot look up the requested name, as there is no name service manager defined.

lcaErrLnsDsNameTooLong **425**

The name specified is too long.

lcaErrLnsDsCountMismatch **426**

Reference/Lock count mismatch. This could be caused by too many Release calls being made by the application.

lcaErrLnsDsNotPermanentMonitorSet **428**

The requested operation can only be performed on a permanent monitor set.

lcaErrLnsDsNotPermanentMonitorPoint **429**

The requested operation can only be performed on a permanent monitor point.

lcaErrLnsDsException **430**

An unexpected exception occurred.

lcaErrLnsDsInitUpdateFailed **431**

A network variable could not be updated.

Formatter Errors

These are errors returned by the LNS formatter. These errors may be returned when LNS fails to read data from a resource file properly. These errors will appear in the format *Subsystem*: Formatter, #<Error Number>. The LNS values for these errors are equal to the error number plus 18,500.

lcaErrFormatNotFound **1**

Not found.

lcaErrFormatNotImplemented **2**

Not implemented.

lcaErrFormatInvalidLocaleData **3**

Invalid locale data.

lcaErrFormatTypeNotFound **4**

Type not found.

lcaErrFormatInvalidTypFile **5**

Invalid type file.

lcaErrFormatReferenceEnumScopeNotFound **6**

Reference enumeration scope not found.

lcaErrFormatTypeEntryNotFound **7**

Type entry not found.

lcaErrFormatFieldNotFound **8**

Field not found.

lcaErrFormatIllegalBaseType **9**

Format specifier is not valid for this type or field.

lcaErrFormatAchNotFound **10**

Format file not found.

lcaErrFormatStringIllegal **11**

Illegal string.

lcaErrFormatStringExceedsBounds **12**

Array index exceeds array size.

lcaErrFormatSpecNotFound **13**

Format specification not found.

lcaErrFormatBuildAchFailed **14**

Compilation of format file failed.

lcaErrFormatIllegalDataOnFormat **15**

Illegal data on format.

lcaErrFormatIllegalDataOnUnformat **16**

Illegal data on unformat.

lcaErrFormatNoUnformat **17**

No format.

lcaErrFormatExceedsSize **18**

Exceeds size.

lcaErrFormatIllegalEnum **19**

Illegal enumeration.

lcaErrFormatExceedsMaxNumFields **20**

Exceeds maximum number of fields.

lcaErrFormatNoResource **21**

No resource.

lcaErrFormatInvalidBuiltInType **22**

Invalid built-in type.

lcaErrFormatNotBuiltInSpec **23**

Not built in spec.

lcaErrFormatInvalidPropSet **24**

Invalid prop set.

lcaErrFormatInvalidBaseType **25**

Invalid base type.

lcaErrFormatFmtNotFound **26**

Format not found.

lcaErrFormatInvalidDefaultType **27**

Invalid default type.

lcaErrFormatUnknownError **28**

Unknown error.

lcaErrFormatFileNotFoundInCatalog **29**

File not found in catalog.

lcaErrFormatCatalogNotOpened **30**

Catalog not opened.

lcaErrFormatEnumNotFound **31**

Enumeration not found.

lcaErrFormatTypeNotOpened **32**

Type not opened.

lcaErrFormatNotVersion2TypeName **33**

Not version 2 type name.

lcaErrFormatInvalidParameter **34**

Invalid parameter.

lcaErrFormatInvalidBfSize **35**

Invalid buffer size.

lcaErrFormatInvalidBfOffset **36**

Invalid buffer offset.

lcaErrFormatEnumNonuniqueSubstr **37**

Substring match was not unique.

lcaErrFormatRawInputTooSmall **38**

Raw input too small.

lcaErrFormatNoFormatFileForTypeFile **39**

No format file for type file.

lcaErrFormatStringDuplicate **40**

Duplicate string provided. This exception will be thrown if you write to the *CategoryPreferenceList* property and specify the same format more than once.

lcaErrFormatLocaleStringNotFound **41**

Locale string not found.

lcaErrFormatCatalogNotFound **200**

Catalog not found.

lcaErrFormatInvalidProgramId **201**

Invalid program ID.

lcaErrFormatLdrfErr **202**

LDRF error.

lcaErrFormatLdrfErrParam **203**

LDRF error: bad parameter.

lcaErrFormatLdrfErrFileType **204**

LDRF error: bad file type.

lcaErrFormatLdrfErrCrc **205**

LDRF error: bad CRC.

lcaErrFormatLdrfErrNotFound **206**

LDRF error: not found.

lcaErrFormatLdrfErrFileInfo **207**

LDRF error: bad file info.

lcaErrFormatLdrfErrSys **208**

LDRF error: sys.

lcaErrFormatLdrfErrTrunc **209**

LDRF error: trunc.

lcaErrFormatLdrfErrStale **210**

LDRF error: stale.

lcaErrFormatLdrfErrVersion **211**

LDRF error: bad version.

lcaErrFormatLdrfErrNew **212**

LDRF error: new.

lcaErrFormatLdrfErrWrite **213**

LDRF error: write failed.

lcaErrFormatLdrfErrNoAccess **214**

LDRF error: no access.

lcaErrFormatLdrfErrFull **215**

LDRF error: full.

lcaErrFormatLdrfErrDuplicate **216**

LDRF error: duplicate.

lcaErrFormatLdrfErrNotCatalog **217**

LDRF error: not catalog.

lcaErrFormatLdrfErrNotResource **218**

LDRF error: not resource.

lcaErrFormatLdrfErrNotType **219**

LDRF error: not type.

lcaErrFormatLdrfErrNotFpt **220**

LDRF error: not FTP.

lcaErrFormatLdrfErrNotFormat **221**

LDRF error: not format.

lcaErrFormatLdrfErrTypeTree **222**

LDRF error: bad type tree.

lcaErrFormatLdrfErrIncomplete **223**

LDRF error: incomplete.

lcaErrFormatLdrfErrSequence **224**

LDRF error: sequence.

lcaErrFormatLdrfErrNotSelected **225**

LDRF error: not selected.

lcaErrFormatLdrfErrInternal **226**

LDRF error: internal error.

lcaErrFormatObjReadOnly **227**

Read-only object.

VNI Errors

VNI errors will be returned when an operation fails while using the *MyVNI* property. These errors will appear in the format VNI: #<Error Number>. The LNS values for these errors are equal to the error number plus 19,000.

lcaErrRmoMaxObjectsAllocated **1**

Maximum objects allocated.

lcaErrRmoBadObjectId **2**

Bad object ID.

lcaErrRmoProcessTimeout **3**

Process timeout.

lcaErrRmoNoServerObject **4**

Remote object does not exist.

lcaErrRmoNoIpcMsgAvail **5**

No Windows Inter Process Communication (IPC) message available.

lcaErrRmoIpcUnknownError **6**

Unknown Windows Inter Process Communication (IPC) error.

lcaErrRmoIpcMsgError **7**

Windows Inter Process Communication (IPC) messaging error.

lcaErrRmoIpcResourceProblem **8**

Windows Inter Process Communication (IPC) resource problem.

lcaErrRmoIpcOutOfMemory **9**

Windows Inter Process Communication (IPC) subsystem is out of memory.

lcaErrRmoIpcOutOfRange **10**

Windows Inter Process Communication (IPC) parameter is out of range.

lcaErrRmoIpcCantFindObject **11**

Windows Inter Process Communication (IPC) subsystem cannot find specified object.

lcaErrRmoIpcLockFailure **12**

Windows Inter Process Communication (IPC) subsystem lock failure.

lcaErrRmoCantLinkToObject **13**

Cannot link to remote object.

lcaErrLtaNoError **300**

LonTalk Adapter error.

lcaErrLtaInvalidParameter **301**

LonTalk Adapter invalid parameter.

lcaErrLtaNotQualified **302**

LonTalk Adapter not qualified.

lcaErrLtaMessageBlocked **303**

LonTalk Adapter message blocked**.**

lcaErrLtaMessageDeferred **304**

LonTalk Adapter message deferred.

lcaErrLtaAppMessage **305**

LonTalk Adapter application message.

lcaErrLtaFlexDomain **306**

LonTalk Adapter flex domain.

lcaErrLtaNoMessage **307**

LonTalk Adapter no message.

lcaErrLtaAppNameTooLong **308**

LonTalk Adapter name too long.

lcaErrLtaInvalidState **309**

LonTalk Adapter invalid state.

lcaErrLtaNoResources **310**

LonTalk Adapter no resources.

lcaErrLtaDuplicateObject **311**

LonTalk Adapter duplicate object.

lcaErrLtaNotImplemented **312**

LonTalk Adapter not implemented.

lcaErrLtaEndOfEnumeration **313**

LonTalk Adapter end of enumeration.

lcaErrLtaOwnerDoesNotExist **314**

LonTalk Adapter owner does not exist.

lcaErrLtaInvalidIndex **315**

LonTalk Adapter invalid index.

lcaErrLtaCantOpenPort **316**

LonTalk Adapter cannot open port.

lcaErrLtaNotFound **317**

LonTalk Adapter not found.

lcaErrLtaNoWinsockDll **318**

LonTalk Adapter no Winsock DLL.

lcaErrLtaCantOpenIpLink **319**

LonTalk Adapter cannot open IP link.

lcaErrLtaCantStartSnmp **320**

LonTalk Adapter cannot start SNMP.

lcaErrLtaNoLin**k** **321**

LonTalk Adapter no IP link.

lcaErrLtaInvalidIpaddress **322**

LonTalk Adapter invalid IP address.

lcaErrLtaLocalMsgFailure **323**

LonTalk Adapter local message failure.

lcaErrLtaStaleNvIndex **324**

LonTalk Adapter stale network variable index.

lcaErrLtaInvalidAddress **325**

LonTalk Adapter invalid address.

lcaErrLtaErrorLogMask **428**

LonTalk Adapter error log mask.

lcaErrLtaNvLengthMismatch 430

LonTalk Adapter network variable length mismatch.

lcaErrLtaNvMsgTooShort 431

LonTalk Adapter network variable message too short.

lcaErrLtaEepromWriteFailure 432

LonTalk Adapter EEPROM write failure.

lcaErrLtaBadAddressType 433

LonTalk Adapter bad address.

lcaErrLtaInvalidDomain 438

LonTalk Adapter invalid domain.

lcaErrLtaInvalidAddrTableIndex 441

LonTalk Adapter invalid address table index.

lcaErrLtaNvUpdateOnOutputNv 443

LonTalk Adapter update on output network variable.

lcaErrLtaUnknownPdu 446

LonTalk Adapter unknown PDU.

lcaErrLtaInvalidNvIndex 447

LonTalk Adapter invalid network variable index.

lcaErrLtaBadError**No 449**

LonTalk Adapter bad buffer number.

lcaErrLtaNetBufTooSmall 451

LonTalk Adapter network buffer too small.

lcaErrLtaCnfgCsError 453

LonTalk Adapter configuration checksum error.

lcaErrLtaXcvrRegOpFailure 455

LonTalk Adapter transceiver register operation failed.

lcaErrLtaSubnetPartition 459

LonTalk Adapter subnet partition.

lcaErrLtaAuthenticationMismatch 460

LonTalk Adapter authentication mismatch.

lcaErrVniUnknownRegError 601

Unknown registry error.

lcaErrVniRegBadParms 602

Bad parameter (registry subsystem).

lcaErrVniRegOutOfMemory 603

Out of memory (registry subsystem).

lcaErrVniRegCantFindObject **604**

Cannot find object (registry subsystem).

lcaErrVniRegResourceProblem **605**

Resource problem (registry subsystem).

lcaErrVniRegOutOfRange **606**

Parameter out of range (registry subsystem).

lcaErrVniRegNoMoreEntries **607**

No more entries (registry subsystem).

lcaErrVniRegCantOpenRegistry **608**

Cannot open registry.

lcaErrVniRegUnknownRegistryError **609**

Unknown error (registry subsystem).

lcaErrVniVniDoesNotExist **610**

Specified VNI does not exist.

lcaErrVniCantCreateVniProcess **611**

Cannot create the remote process.

lcaErrVniCantLockVniServerDir **612**

Cannot lock the server directory.

lcaErrVniOpenObjectConflict **613**

Tried to use two different objects to represent the same VNI object.

lcaErrVniOpenFailure **614**

VNI open failure.

lcaErrVniNoMoreMonitorPoints **615**

No more monitor points found.

lcaErrVniNoMoreMonitorSets **616**

No more monitor sets found.

lcaErrVniStackNotOpen **617**

VNI stack is not open.

lcaErrVniNotImplemented **618**

Not implemented.

lcaErrVniMonitorSetNotFound **619**

Monitor set not found.

lcaErrVniNvPointNotFound **620**

Network variable monitor point not found.

lcaErrVniMsgPointNotFound **621**

Message monitor point not found.

lcaErrVniMustSpecifyMsgTag 622

Must specify message tag.

lcaErrVniMsgTagNotFound 623

Message tag not found.

lcaErrVniOutOfTemporaryMonitorPoints 624

Monitor set ran out of temporary monitor points due to capacity constraints.

lcaErrVniOutOfTemporaryMonitorSets 625

VNI stack ran out of temporary monitor sets due to capacity constraints.

lcaErrVniAddPointToPersistentMonitorSet 626

Attempted to add a point to a persistent monitor set.

lcaErrVniDelPointFromPersistentMonitorSet 627

Attempted to delete a point from a persistent monitor set.

lcaErrVniAbosoluteRegPathNotSupported 628

Absolute registry paths not supported.

lcaErrVniRegBadCollectionPath 629

Cannot define a collection of VNIs at this level in the registry.

lcaErrVniOpenStackNiMismatch 630

The VNI is already open with different network interface settings.

lcaErrVniTraceFileCantBeOpened 631

The specified trace file cannot be opened.

lcaErrVniNoMoreMessageBuffers 632

The client has more messages outstanding than permitted.

lcaErrVniIsNotAnIpDevice 633

The specified device is not an IP device.

lcaErrVniL5MipInUse 634

The requested operation is not supported while a Layer 5 network interface is in use.

lcaErrVniXifCannotBeFound 635

The external interface file cannot be found.

lcaErrVniXifReadError 636

Failed to read external interface file.

lcaErrVniAllocation 637

VNI allocated error.

lcaErrVniProgramInterfaceMismatch 638

Program interface mismatch.

lcaErrVniRegConfigFileNotFound 639

Configuration file could not be found.

lcaErrVniRegCannotUpdateConfigFile 640

Configuration file could not be updated.

lcaErrVniThreadCreationFailure 641

Failed to create thread.

lcaErrVniDumpLtipXmlConfigFailed 642

XML configuration failed.

lcaErrVniNotSupportedOnL5Mip 643

The requested operation is not supported on a Layer 5 network interface.

lcaErrVniInvalidMessagePointOptions 644

Invalid message point options.

Appendix A
Deprecated Items

This appendix lists methods, properties, and objects that should no longer be used in OpenLNS.

Deprecated Items

In OpenLNS and LNS Turbo Editions, some methods, objects, properties and events have been deprecated. This section provides a list of those items. Some have been deprecated because they were never implemented in LNS, or because they are no longer applicable or useful. Others have been deprecated because they have been replaced by new features with better functionality.

Note that some of the objects, methods, properties and events marked as deprecated in the documentation are still implemented in OpenLNS to maintain backwards-compatibility with applications running on previous versions of LNS. For example, many of the methods of the *ConfigProperty* object were deprecated in LNS Turbo Editions as a result of the new data point feature, including the *GetElement()*, *GetElementFromDevice()*, *SetElement()*, and *SetElementFromDevice()* methods. You can still successfully use these methods when running on OpenLNS; however, Echelon recommends that you use the *DataPoint* object to read and write configuration properties, and so these methods have been marked deprecated.

The following sections list the deprecated objects, properties, methods and events in OpenLNS. The reason for deprecation can be determined using the following codes:

BA – Better Feature Available. The feature is still implemented in LNS for compatibility purposes, but a more efficient way to achieve its purpose exists. Items marked with this reason include details on why the item was deprecated and the features to use in its place.

U – Unimplemented. The feature was never implemented in LNS.

NLA – No Longer Applicable. The feature is still implemented in LNS for compatibility purposes, but is no longer useful or functional because of other changes to the LNS implementation.

Deprecated Objects

The following objects have been deprecated in OpenLNS and LNS Turno Editions (3.2). This means that you should no longer use the object, or any of its properties and methods, in your applications.

Object	**Version**	*Reason*	**Substitute Feature**
BuildTemplate	3.2	U	
BuildTemplates	3.2	U	
HardwareTemplate	3.2	U	
HardwareTemplates	3.2	U	
NetworkVariableField	3.2	BA	Use *DataPoint* objects to read and write the values of the network variables and network variable fields on your network. You can use the *GetDataPoint* method to acquire a *DataPoint* for reading or writing the value of a network variable. If the network variable is a structure, you can use the *GetField* method to access any of the fields of the source network variable once you have used the *GetDataPoint* method to acquire a *DataPoint* for a network variable.
ProgramTemplate	3.2	U	
ProgramTemplates	3.2	U	

Deprecated Methods

The following methods have been deprecated in OpenLNS and LNS Turbo Editions (3.2). This means that you should no longer use these methods in your applications. Note that the parent objects of these methods have not been deprecated unless those objects are listed in the *Deprecated Objects* section.

Method	**Applicable Parent Object**	**Version**	*Reason*	**Substitute Feature**
BeginLicense	*System*	4.0	NLA	
Build	*AppDevice* *System*	3.2	U	
CloseComponent	*AppDevice*	3.2	U	
DebitLicense	*System*	4.0	NLA	
EndLicense	*System*	4.0	NLA	
Export	*AppDevice* *DeviceTemplate*	3.2	U	
GetElement *GetElementFromDevice*	*ConfigProperty*	3.2	BA	Use *DataPoint* objects to read and write to the values of all configuration properties and configuration property arrays. You can use the *GetDataPoint* method of the *ConfigProperty* object to begin this procedure.
GetField	*NetworkVariable*	3.2	BA	If the network variable is a structure, you can use the *GetField* method to access any of the fields of the source network variable once you have used the *GetDataPoint* method to acquire a *DataPoint* for a network variable.
GetRawValues *GetRawValuesFromDevice*	*ConfigProperty*	3.2	BA	Use the *GetRawValuesEx* method to read the raw values of configuration property arrays.
Link	*DeviceTemplate*	3.2	U	
Lock	*System*	3.2	U	

Method	Applicable Parent Object	Version	*Reason*	Substitute Feature
OpenComponent	*AppDevice*	3.2	U	
Purge	*Interface*	3.2	BA	Use the *RemoveNv* method to remove network variables from an *Interface* object.
RecoverFromNssDb	*System*	3.2	U	
Remove	*Networks*	3.2	BA	Use the *RemoveEx* method to remove a network from a *Networks* collection.
RestoreLicense	*System*	4.0	NLA	
SetCapacity	*System*	3.2	NLA	
SetCustomerInfo	*ObjectServer*	4.0	NLA	
SetElement *SetElementFromDevice*	*ConfigProperty*	3.2	BA	Use *DataPoint* objects to read and write to the values of all configuration properties and configuration property arrays. You can use the *GetDataPoint* method of the *ConfigProperty* object to begin this procedure.
SetLicenseInfo	*ObjectServer*	4.0	NLA	
SetLicenseInfoEx	*ObjectServer*	4.0	NLA	
SetRawValues *SetRawValuesFromDevice*	*ConfigProperty*	3.2	BA	Use the *SetRawValuesEx* method to write to the raw values of configuration property arrays.
Unlock	*System*	3.2	U	

Deprecated Properties

The following propeties have been deprecated in OpenLNS and LNS Turbo Editions (3.2). This means that you should no longer use these properties in your applications. Note that the parent objects of these methods have not been deprecated unless those objects are listed in the *Deprecated Objects* section.

Property	**Applicable Parent Object**	**Version**	*Reason*	**Substitute Feature**
ActiveXComponent	*AppDevice*	3.2	U	
BuildStatus	*AppDevice* *DeviceTemplate* ProgramTemplate	3.2	U	
BuildTemplate	ProgramTemplate *System*	3.2	U	
BuildTemplates	*TemplateLibrary*	3.2	U	
CompatibleNv	*NetworkVariable*	3.2	BA	Use the Add method of the *NetworkVariables* collection to create new network variables.
ComplementaryNv	*NetworkVariable*	3.2	BA	Use the Add method of the *NetworkVariables* collection to create new network variables.
ConnErrNvMtIndex1	Error	3.2	BA	Use the *ConnErrIndex1* property, which supports Long index numbers. The *ConnErrNvMtIndex1* property is currently included in LNS for backwards compatibility only.
ConnErrNvMtIndex2	Error	3.2	BA	Use the *ConnErrIndex2* property, which supports Long index numbers. The *ConnErrNvMtIndex2* property is currently included in LNS for backwards compatibility only.
DataServerHandle	*System*	3.2	NLA	

Property	**Applicable Parent Object**	**Version**	*Reason*	**Substitute Feature**
DataServerObjectHandle	*System*	3.2	NLA	
DaysRemaining	*System*	4.0	NLA	
DefecitCredits	CreditInfo	4.0	NLA	
DsAuthenticate	*NetworkVariable* NetworkVariableField	3.2	BA	Use temporary monitor sets instead of single-point monitoring. You can use the *Authentication* property of the *MsgMonitorOptions* and *NvMonitorOptions* objects to determine the authentication setting that will be applied to temporary monitor points. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.
DsAutoUpdate	*NetworkVariable*	3.2	U	
DsEventSubscription	*System*	3.2	NLA	
DsFormatFilesPath	*System*	3.2	NLA	
DsMessageOwner	*System*	3.2	NLA	
DsMode	*System*	3.2	NLA	
DsMonitorTag	*NetworkVariable* NetworkVariableField	3.2	BA	Use temporary monitor sets instead of single-point monitoring. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.
DsPause	*NetworkVariable* NetworkVariableField	3.2	NLA	
DsPrecision	*NetworkVariable* NetworkVariableField *System*	3.2	BA	Use the *FloatPrecision* property of the *FormatLocale* object being used by your application to establish the precision to use

Property	**Applicable Parent Object**	**Version**	*Reason*	**Substitute Feature**
				when displaying network variable values.
DsPriority	*NetworkVariable* NetworkVariableField	3.2	BA	Use temporary monitor sets instead of single-point monitoring. You can use the *Priority* property of the *MsgMonitorOptions* and *NvMonitorOptions* objects to determine the priority assigned to each message. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.
DsReportByException	*NetworkVariable* NetworkVariableField *System*	3.2	BA	Use temporary monitor sets instead of single-point monitoring. You can use the *ReportByException* property of the *MsgMonitorOptions* and *NvMonitorOptions* objects to determine when update events will be generated. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.
DsRetries	*NetworkVariable*	3.2	BA	Use temporary monitor sets instead of single-point monitoring. You can use the *Retries* property of the *MsgMonitorOptions* and *NvMonitorOptions* objects to determine the retry count assigned to each monitor point. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.

Property	**Applicable Parent Object**	**Version**	*Reason*	**Substitute Feature**
DsService	*NetworkVariable* NetworkVariableField *System*	3.2	BA	Use temporary monitor sets instead of single-point monitoring. You can use the *ServiceType* property of the *MsgMonitorOptions* and *NvMonitorOptions* objects to determine the service type to apply to the temporary monitor points. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.
DsUseBoundUpdates	*NetworkVariable* NetworkVariableField	3.2	BA	To enable bound updates for a network variable, create a monitor point for the network variable, and set the *UseBoundUpdates* property of the *MsgMonitorOptions* and *NvMonitorOptions* objects to True. For more information on temporary monitor sets, see Chapter 9 of the *OpenLNS Programmer's Guide*.
DynamicNvPersistenceMode	*System*	3.2	BA	Use the *Remove* method to remove network variables from a *NetworkVariables* collection object.
ExportDirectory	*System*	3.2	U	
ExportFormat	*DeviceTemplate*	3.2	U	
GraphicsDirectory	*System*	3.2	U	
HardwareTemplate	*DeviceTemplate*	3.2	U	
HardwareTemplates	*TemplateLibrary*	3.2	U	
LicenseCredits	CreditInfo	4.0	NLA	
LicenseType	CreditInfo	4.0	NLA	
LockDuration	*System*	3.2	U	

Property	**Applicable Parent Object**	**Version**	*Reason*	**Substitute Feature**
MaxDefecitCredits	CreditInfo	4.0	NLA	
MaximumDeviceCapacity	*System*	4.0	NLA	
ProgramTemplate	*DeviceTemplate*	3.2	U	
ProgramTemplates	*TemplateLibrary*	3.2	U	
RawValue *RawValueFromDevice*	*ConfigProperty*	3.2	BA	Use *DataPoint* objects to read and write to the values of all configuration properties and configuration property arrays. You can use the *GetDataPoint* method of the *ConfigProperty* object to begin this procedure.
RemoteIgnorePendingUpdate	*Network*	3.2	BA	Use the *AllowPropagateModeDuringRemoteOpen* property.
SingleUserMode	*ObjectServer*	3.2	NLA	
TypeDefaultValue	*ConfigProperty*	3.2	BA	Use *DataPoint* objects to read the default values of the *ConfigProperty* objects on your network. You can use the *GetDataPoint* method to acquire a *DataPoint* for reading the default value of a network variable. When you do this, you need to specify **lcaDataSourceOptionsTypeDefaultValue** (3) as the options element.
UsedCredits	CreditInfo	4.0	NLA	
Value	*ConfigProperty*	3.2	BA	Use *DataPoint* objects to read and write the value of the *ConfigProperty* objects on your network. You can use the *GetDataPoint* method to acquire a *DataPoint* for reading or writing the value of a configuration property.

Property	Applicable Parent Object	Version	*Reason*	Substitute Feature
ValueFromDevice	*ConfigProperty*	3.2	BA	Use *DataPoint* objects to read and write the value of the *ConfigProperty* objects on your network. You can use the *GetDataPoint* method to acquire a *DataPoint* for reading or writing the value of a configuration property.

Deprecated Events

The following events have been deprecated in OpenLNS and LNS Turno Editions (3.2). This means that you should no longer use the object, or any of its properties and methods, in your applications.

Event	Applicable Parent Object	Version	*Reason*	Substitute Feature
OnBuildMessage	*ObjectServer*	3.2	U	
OnLicenseEvent	*ObjectServer*	4.0	NLA	
OnNetworkServiceDeviceReset	*ObjectServer*	3.2	NLA	

ECHELON®
www.echelon.com