



*i.LON*® SmartServer 2.0  
Programming Tools User's Guide



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

Other brand and product names are trademarks or registered trademarks of their respective holders.

Neuron Chips, LonPoint Modules, and other OEM Products were not designed for use in equipment or systems which involve danger to human health or safety or a risk of property damage and Echelon assumes no responsibility or liability for use of the Neuron Chips or LonPoint Modules in such applications.

Parts manufactured by vendors other than Echelon and referenced in this document have been described for illustrative purposes only, and may not have been tested by Echelon. It is the responsibility of the customer to determine the suitability of these parts for each application.

ECHELON MAKES NO REPRESENTATION, WARRANTY, OR CONDITION OF ANY KIND, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE OR IN ANY COMMUNICATION WITH YOU, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR ANY PARTICULAR PURPOSE, NONINFRINGEMENT, AND THEIR EQUIVALENTS.

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

Printed in the United States of America.  
Copyright ©1997–2009 by Echelon Corporation.  
Echelon Corporation  
[www.echelon.com](http://www.echelon.com)

---

# Table of Contents

<b>Preface</b> .....	<b>viii</b>
Welcome .....	ix
Purpose .....	ix
Audience .....	ix
Models .....	ix
i.LON SmartServer 2.0 Programming Tools Versions .....	ix
i.LON SmartServer 2.0 Programming Tools Applications .....	ix
Hardware Requirements .....	x
SmartServer Requirements .....	x
Creating FPM Application Licenses .....	xi
i.LON SmartServer 2.0 Documentation .....	xi
Related Reading .....	xi
Content .....	xii
For More Information and Technical Support .....	xiii
<b>1 Introduction</b> .....	<b>1</b>
Overview of Freely Programmable Modules .....	2
FPM Types .....	2
Creating and Deploying FPMs .....	2
Using Eclipse Environment Commands .....	3
Debugging FPMs .....	4
Creating FPM Application Licenses .....	5
Quick-Start FPM Exercise .....	6
Step 1: Creating and Copying the FPM Template .....	7
Step 2: Creating and Copying the Device Interface (XIF) File .....	8
Step 3: Creating the FPM Project .....	9
Step 4: Writing the FPM Application .....	12
Step 5: Deploying the FPM Application on a SmartServer .....	13
Uploading the FPM Application .....	13
Creating an Internal FPM device .....	15
Step 6: Testing the FPM Application .....	17
Step 7: Connecting the FPM Data Points .....	17
<b>2 Installing i.LON SmartServer 2.0 Programming Tools</b> .....	<b>23</b>
Installation and Upgrading Overview .....	24
Installing i.LON SmartServer 2.0 Programming Tools .....	24
Upgrading the i.LON SmartServer 2.0 Programming Tool .....	29
Importing FPM Projects .....	30
Converting FPM Projects to the Release 4.03 Configuration .....	33
Uninstalling i.LON SmartServer 2.0 Programming Tools .....	39
<b>3 Creating FPM Templates</b> .....	<b>41</b>
Creating FPM Templates Overview .....	42
Creating User-Defined Functional Profile Templates .....	42
Adding Network Variable and Configuration Property Types .....	48
Generating and Copying the Updated FPM Resource File Set .....	53
<b>4 Creating FPM Device Interface (XIF) Files</b> .....	<b>55</b>
Creating FPM Device Interface (XIF) Files Overview .....	56
Creating a Model File .....	56
Declaring Network Variables .....	56
Declaring Configuration Properties .....	57
Declaring Functional Blocks .....	58

Using Include Directives .....	59
Example Model Files .....	59
Saving your Model File .....	62
Generating a Device Interface (XIF) File .....	63
Using Long and Short Command Switch Forms .....	64
Other Command Switches.....	64
<b>5 Creating FPMs .....</b>	<b>65</b>
Creating FPMs Overview.....	66
Creating New FPM Projects.....	67
Viewing the Resource Files on a SmartServer.....	67
Creating an FPM .....	68
Updating Data Point Declarations .....	72
Using UFPT Local Variables.....	75
Writing an FPM Application.....	76
The Writing the FPM Application Initialize() Routine .....	76
Writing the FPM Application Work() Routine.....	78
Writing the FPM Application OnTimer() Routine .....	84
Writing the FPM Application Shutdown() Routine .....	85
Writing an FPM Driver.....	86
Writing the FPM Driver Initialize() Routine .....	86
Writing the FPM Driver Work() Routine.....	87
Writing the FPM Driver OnTimer() Routine .....	87
Writing the FPM Driver Shutdown() Routine .....	88
Compiling an FPM.....	89
Checking Compile and Warning Errors .....	89
Using Non-Latin Characters .....	90
Debugging FPMs .....	91
Using Wind River Workbench .....	92
Using FPM Development Guidelines .....	101
Using SNMP Support.....	102
Example FPM Applications and Drivers.....	102
<b>6 Deploying FPMs on a SmartServer.....</b>	<b>105</b>
FPM Deployment Overview.....	106
Uploading FPM Applications and Drivers .....	107
Deploying FPM Applications.....	111
Deploying FPM Drivers.....	111
Selecting a Network Management Service.....	112
Using LNS Network Management Services .....	112
Using Standalone Network Management.....	113
Adding FPM Devices to the SmartServer .....	114
Using a Static Device Interface .....	114
Using a Dynamic Device Interface .....	117
Commissioning FPM Devices .....	120
Commissioning FPM Devices with the SmartServer.....	120
Commissioning FPM Devices with the LonMaker Tool.....	120
Recommissioning FPM Devices.....	121
Testing FPM Applications .....	122
Connecting FPM Data Points.....	122
Creating LONWORKS Connections .....	123
Creating Web Connections.....	127
Creating Custom FPM Configuration Web Pages .....	133
Updating FPMs .....	138
Updating Data Point Declarations .....	138
Updating FPM Applications and Drivers.....	139

Updating Device Interfaces .....	139
Deploying FPMs on Multiple SmartServers.....	142
Deploying Licensed FPM Applications .....	143
<b>7 Creating FPM Application Licenses .....</b>	<b>145</b>
Licensing Overview .....	146
Creating an FPM Licensing Tool.....	146
Creating a License Generator Configuration File.....	146
Creating a Security DLL File.....	149
Enabling License Validation in an FPM Application.....	150
Step 1: Inserting Include Directives and Macro Definitions.....	152
Step 2: Declaring Data Variables .....	153
Step 3: Creating the License Validation Routine.....	154
Step 4: Writing the License Validation Algorithm .....	156
Step 5: Implementing the License Validation Call Mechanism .....	160
Step 6: Compiling the Licensed FPM Application .....	160
Building the Release Version of a Licensed FPM Application.....	160
Creating FPM Application Licenses.....	161
Supplying FPMs to Customers .....	164
<b>8 Localizing the SmartServer Web Interface .....</b>	<b>167</b>
Language Localization Overview .....	168
Creating a Language Localization Project.....	168
Creating Localized Custom SmartServer Web Pages.....	172
Translating Common Properties.....	173
Translating Embedded Application Properties .....	178
Creating a Localized Custom SmartServer Web Page .....	179
Creating Localized FPM Configuration Web Pages .....	182
Localizing the Language of the SmartServer Web Interface .....	185
Translating Property Files.....	185
Creating New Language Folders.....	185
Editing the index.htm File to Enable a New Language on the SmartServer .....	186
Translating the Welcome.htm File.....	187
Translating the Menu.htm File .....	191
Translating the Sidebar.htm File .....	198
Viewing the Localized SmartServer Web Interface.....	201
<b>Appendix A FPM Programmer's Reference.....</b>	<b>203</b>
Overview.....	204
Template Files .....	204
Routines.....	204
Initialize() .....	205
FPM Application Example .....	205
FPM Driver Example .....	205
Work() .....	205
FPM Application Example .....	206
FPM Driver Example .....	206
OnTimer().....	206
FPM Application .....	207
FPM Driver.....	207
Shutdown().....	208
FPM Application Example .....	208
FPM Driver Example .....	208
Methods.....	208
Variable Types .....	208

Internal FPM Data Point Methods.....	209
Changed().....	209
NotifyOnAllUpdates().....	210
Propagate().....	210
Write().....	211
ResetPriority().....	211
FPM Application Data Point Property Methods .....	212
GetDpPropertyAsString(UCPTname).....	212
GetDpPropertyAsString(UCPTAliasName) .....	213
GetDpPropertyAsTimeSpec(UCPTlastUpdate).....	213
GetDpPropertyAsPointStatus(UCPTstatus) .....	214
GetDpPropertyAsInt(UCPTpriority) .....	215
SetDpProperty (UCPTAliasName) .....	215
SetDpProperty (UCPTpriority).....	216
FPM Driver Data Point Property Methods .....	216
SetDpProperty(defOutput).....	216
SetDpProperty(persist) .....	217
SetDpProperty(pollRate) .....	217
SetDpProperty(unit).....	217
UFPT Local Variables .....	217
External SmartServer Data Point Methods .....	218
List().....	218
Read().....	219
Write() .....	220
Timer Methods .....	221
Start().....	221
START_TIMER() .....	221
Expired() .....	222
Stop () .....	223
StopAllTimers().....	223
IsRunning() .....	223
GetMode() .....	224
GetTimeoutMillis() .....	224
Reboot Method.....	224
RS-232 Interface Methods .....	225
rs232_open() .....	225
rs232_ioctl().....	226
rs232_read() .....	227
rs232_write().....	228
rs232_close().....	229
RS-485 Interface Methods .....	229
rs485_open() .....	229
rs485_setparams() .....	230
rs485_ioctl().....	230
rs485_setbuffersize().....	231
rs485_read() .....	231
rs485_write().....	232
rs485_close().....	232
File Access Methods.....	233
fopen() .....	233
fread() .....	234
fseek().....	235
fwrite().....	235
fclose().....	236

**Appendix B FPM Development and Deployment Checklist ..... 237**

**Appendix C FPM FAQ ..... 241**

# Preface

You can use *i.LON SmartServer 2.0 Programming Tools* to create custom embedded applications and drivers, which are referred to as freely programmable modules (FPMs), for your SmartServer. FPMs let you customize the embedded software of the SmartServer to meet your specific needs. Using *i.LON SmartServer 2.0 Programming Tools*, you can write FPMs in C or C++, compile them, and then upload them to your SmartServer. You can then deploy your FPMs on SmartServers that have an FPM programming license installed on them. You can also create FPM application licenses and use them to protect your FPM applications and make them available to customers for order.



---

## Welcome

The SmartServer includes *i.LON* SmartServer 2.0 Programming Tools for creating custom C/C++ applications and drivers (called freely programmable modules [FPMs]) that you can use to customize the functionality of the SmartServer. You can use your FPMs for a number of applications, including energy optimization, data analysis, lighting control, and room control. You can also use the *i.LON* SmartServer 2.0 Programming Tools to translate the SmartServer Web interface into a number of different languages (language localization).

---

## Purpose

This guide describes how to create and use FPMs on your SmartServer, and how to localize the language of the SmartServer Web interface.

---

## Audience

This guide is intended for system designers and integrators with an understanding of control networks and the ability to program in C or C++, and for language localization developers.

---

## Models

This guide is intended for FT-10 and PL-20 models of the SmartServer hardware on which FPM Programming is licensed. This includes models of the SmartServer on which the FPM programming license is pre-installed (model numbers 72101R-439, 72101R-440, 72102R-445, 72103R-439, and 72103R-445), and all other models of the SmartServer hardware for which the FPM programming license (Echelon part number 72161) has been ordered and installed.

---

## *i.LON* SmartServer 2.0 Programming Tools Versions

The *i.LON* SmartServer 2.0 DVD includes a demo version of the *i.LON* SmartServer 2.0 Programming Tools. You can use the demo version to write an unlimited number of FPMs. To compile your FPMs and deploy them on your SmartServer, you must order an *i.LON* SmartServer 2.0 Programming Tools DVD. To order the *i.LON* SmartServer 2.0 Programming Tools DVD (Echelon part number 72111-409), contact your Echelon sales representative.

---

## *i.LON* SmartServer 2.0 Programming Tools Applications

Installing the demo or full version of the *i.LON* SmartServer 2.0 Programming Tools adds the following programs to your computer:

- *i.LON* SmartServer 2.0 Programming Tool. A pre-configured Eclipse Development Kit that includes FPM template files, the FPM library, a tool for creating the C structures of user-defined UNVTs, a C++ compiler, and a CYGWIN environment. You must have the full version of the *i.LON* SmartServer 2.0 Programming Tools to compile and upload FPMs to your SmartServer with the *i.LON* SmartServer 2.0 Programming Tool.
- *i.LON* SmartServer 2.0 LonWorks Interface Developer tool. A command line interface that converts a model file (.nc extension) to a device interface (XIF) file. You must create a XIF for your FPM in order to deploy it on your SmartServer. See Chapter 4 for more information on creating XIFs with this tool.
- *i.LON* License Generator. A tool for creating licenses that help protect your FPM application from piracy or unauthorized use. The *i.LON* License Generator includes the following three components:
  - The main executable (**iLONLicenseGen.exe**) that provides a user interface for entering the values used to generate an FPM license.

- A sample license generator configuration file (an XML file named **iLONLicenseGenValuesSample.xml**) that demonstrates the structure of the *i.LON* License Generator user interface and provides sample pre-defined values.
- A sample security DLL file (**LicenseSecurityHMACMD5.dll**) that takes the values entered in the *i.LON* License Generator user interface and creates an FPM license.

See Chapter 7 for more information on creating FPM application licenses.

---

## Hardware Requirements

Requirements for the running the *i.LON* SmartServer 2.0 Programming Tools are listed below:

- Microsoft® Windows Vista® or Microsoft Windows® XP. Echelon recommends that you install the latest service pack available from Microsoft for your version of Windows.
- Intel® Pentium® IV 1.5GHz processor or faster, and meeting the minimum Windows requirements for the selected version of Windows.
- 1 GB RAM minimum.

**Note:** Windows Vista testing for the *i.LON* SmartServer 2.0 Programming Tools has been performed on computers that have a minimum of 2 GB of RAM. For complete Windows Vista requirements, refer to [www.microsoft.com/windows/windows-vista/get/system-requirements.aspx](http://www.microsoft.com/windows/windows-vista/get/system-requirements.aspx). You can use Microsoft's Vista Upgrade Advisor to determine upgrade requirements for a particular computer. To download this tool, go to the Microsoft Web site at [www.microsoft.com/windows/windows-vista/get/upgrade-advisor.aspx](http://www.microsoft.com/windows/windows-vista/get/upgrade-advisor.aspx).

- 300 megabytes (MB) free hard-disk space, plus the minimum Windows requirements for the selected version of Windows.
- DVD-ROM drive.
- 1024x768 or higher-resolution display with at least 256 colors.
- Mouse or compatible pointing device.
- Microsoft Internet Explorer 7 or higher or Mozilla Firefox.
- Terminal emulator such as Windows HyperTerminal. If you are using Windows Vista, you need to install a terminal emulation application on your computer (Windows HyperTerminal is not included with Windows Vista). You can license Windows HyperTerminal from Hilgraeve; install puTTY, which is included on the root directory of the *i.LON* SmartServer 2.0 DVD and the *i.LON* SmartServer 2.0 Programming Tools DVD; or download another free terminal emulator to your computer.

---

## SmartServer Requirements

You can run FPMs on the SmartServer hardware (they cannot be run on *i.LON* 100 *e3* server hardware). A SmartServer image and an FPM programming license must be installed on the SmartServer hardware.

You can run existing FPMs built with the *i.LON* SmartServer 1.0 Programming Tool (Release 4, Release 4.01, or Release 4.02 FPMs) on a SmartServer 2.0. If you want to upgrade existing FPMs built with the *i.LON* SmartServer 1.0 Programming Tool, you must convert them to the Release 4.03 configuration using the *i.LON* SmartServer 2.0 Programming Tool. After you convert your existing FPMs to the Release 4.03 configuration, you can modify, rebuild, and upload them with the *i.LON* SmartServer 2.0 Programming Tool, and run them on your SmartServer 2.0.

To run your FPMs on your SmartServer, your SmartServer must have an FPM programming license installed on it. If you do not have a SmartServer model that includes a pre-installed FPM programming license, you can order a FPM programming license from the *i.LON* SmartServer 2.0

Web site at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate). To run Echelon first-party FPMs or third-party FPMs on your SmartServer, your SmartServer must also have a separate FPM application license from Echelon or the third-party FPM vendor.

---

## Creating FPM Application Licenses

You can create FPM application licenses for your FPMs to protect your FPMs from unauthorized use and piracy, and to make your FPMs available to customers for order. The *i.LON SmartServer 2.0 Programming Tools* includes the components required to create a FPM licensing tool. Once you create your FPM licensing tool, you can use it to create FPM application licenses that must be installed on a SmartServer in order for it to run your FPMs. Customers who want to implement your licensed FPMs on their SmartServers must order your FPM application license and install it on their SmartServers.

---

## *i.LON SmartServer 2.0* Documentation

The documentation for the SmartServer is provided as Adobe Acrobat PDF files and online help files. The PDF file for this document is installed in the **Echelon *i.LON SmartServer 2.0 Programming Tools*** program folder when you install the *i.LON SmartServer 2.0 Programming Tools* software. You can also download the latest SmartServer documentation, including the latest version of this guide, from Echelon's Website at [www.echelon.com/support/documentation/manuals/cis](http://www.echelon.com/support/documentation/manuals/cis).

This user's guide, the online help files, and the following documents comprise the SmartServer documentation suite:

- *i.LON SmartServer 2.0 User's Guide*. Describes how to configure the SmartServer and use its applications to manage control networks
- *Echelon Enterprise Services 2.0 User's Guide*. Describes how to use the *i.LON AdminServer* to rapidly and automatically deploy and install LONWORKS networks and how to use the LNS Proxy Web service to manage LNS networks.
- *i.LON Vision 2.0 User's Guide*. Describes how to create custom Web pages for monitoring and controlling LONWORKS networks and other control networks.
- *i.LON SmartServer 2.0 Power Line Repeating Network Management Guide*. Describes how to install a PL-20 repeating network and how to use the SmartServer to prepare, maintain, monitor and control, and connect the network.
- *i.LON SmartServer 2.0 Programmer's Reference*. Describes how to configure the SmartServer using XML files and SOAP calls. This allows you to create your own applications that you can use to configure the SmartServer.
- *i.LON SmartServer 2.0 Hardware Guide*. Describes how to assemble, mount, and wire the SmartServer hardware.
- *i.LON SmartServer 2.0 Quick Start Guide*. Contains all the information you will need to connect the SmartServer hardware, install the *i.LON SmartServer 2.0* software, and configure the SmartServer using the SmartServer configuration Web pages.
- *IP-852 Channel User's Guide*. Describes how to configure an IP-852 channel with the Echelon LONWORKS<sup>®</sup>/IP Configuration Server. You will need this information if you plan to use the *i.LON* as an IP-852 router.

---

## Related Reading

The following additional documents may be useful if you are using certain features of the SmartServer. You can download these documents from Echelon's Web site at [www.echelon.com/docs](http://www.echelon.com/docs).

- *NodeBuilder Resource Editor User's Guide*. Describes how to use the NodeBuilder Resource Editor to create and edit functional profile templates.

- *Neuron C Programmer's Guide*. Describes how to write programs using the Neuron<sup>®</sup> C Version 2.1 language.
- *Neuron C Reference Guide*. Provides reference information for writing programs using the Neuron C language.
- *LONMARK Resource Files, version 13.00*. Documents the standard network variable types (SNVTs), standard configuration property types (SCPTs), and standard enumeration types that you can declare in your FPM applications and drivers. You can go to [types.lonmark.org/index.html](http://types.lonmark.org/index.html) to check the current LONMARK standard resource file.

---

## Content

This guide includes the following content:

- *Introduction*. Provides an overview of freely programmable modules (FPMs) and explains the types of tasks FPMs can perform. Describes the types of custom embedded applications and drivers you can create with FPMs. Explains how to create and configure FPMs. Summarizes how to create FPM application licenses in order to protect your FPM applications and make them available to customers for order. Provides a quick-start exercise that you can use to create a simple FPM application.
- *Installing the i.LON SmartServer 2.0 Programming Tools*. Describes how to install, upgrade, and uninstall the i.LON SmartServer 2.0 Programming Tools.
- *Creating FPM Templates*. Describes how to use the NodeBuilder Resource Editor to create the user-defined functional profile templates (UFPTs) to be used by your FPMs. Explains how to upload the UFPTs to your SmartServer so that you can begin writing your FPMs.
- *Creating FPM Device Interface (XIF) Files*. Describes how to create a static device interface (XIF) file for your FPM. This step is required if you are integrating your FPM applications with the LonMaker tool or another LNS network tool. Describes how to write a model file that declares the network variables and configuration properties in your FPM and a functional block implementing an instance of the UFPT used by your FPM. Explains how to use the i.LON SmartServer 2.0 LONWORKS Interface Developer tool to convert your model file to a device interface (XIF) file and how to copy the XIF to your SmartServer.
- *Creating FPMs*. Describes how to use the i.LON SmartServer 2.0 Programming Tool to create a new FPM project, and then write, compile, and debug FPM applications and FPM drivers.
- *Deploying FPMs on a SmartServer*. Describes how to use the i.LON SmartServer 2.0 Programming Tool to upload FPMs to one or more SmartServers. Explains how to select a network management service (LNS or Standalone) for running your LONWORKS network. Describes how to create, commission, and connect, and test FPM devices on the SmartServer. Describes how to create a custom configuration Web page for FPM applications. Explains how to update an FPM application. Describes how to deploy licensed FPMs.
- *Creating FPM Application Licenses*. Describes how to create FPM application licenses so that customers can order and implement your FPMs on their SmartServers. Describes how to build an FPM licensing tool. Explains how to enable license validation in your FPM application. Describes how to create FPM application licenses. Lists the files you need to provide to customers who order your licensed FPM applications.
- *Localizing the SmartServer Web Interface*. Describes how to translate custom SmartServer Web pages and the entire SmartServer Web interface to a different language.
- *Appendices*. Includes a programmer's reference for writing FPM applications and drivers, a checklist outlining the FPM development and deployment process, and an FAQ.

---

## For More Information and Technical Support

The **i.LON SmartServer 2.0 Programming Tools ReadMe** document provides descriptions of known problems, if any, and their workarounds. To view the i.LON SmartServer 2.0 Programming Tools ReadMe document, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then select **i.LON SmartServer 2.0 Programming Tools - ReadMe**. You can also find additional information about the SmartServer at the i.LON Web page at [www.echelon.com/ilon](http://www.echelon.com/ilon).

If you have technical questions that are not answered by this document, the SmartServer 2.0 online help, or the **i.LON SmartServer 2.0 Programming Tools 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](http://www.echelon.com/support) for more information on Echelon support and training services.

You can also view free online training or enroll in training classes at Echelon or an Echelon training center to learn more about developing devices. You can find additional information about device development training at [www.echelon.com/training](http://www.echelon.com/training).

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

<b>Region</b>	<b>Languages Supported</b>	<b>Contact Information</b>
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 <a href="mailto:lonsupport@echelon.com">lonsupport@echelon.com</a>
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 <a href="mailto:lonsupport@echelon.co.uk">lonsupport@echelon.co.uk</a>
Japan	Japanese	Echelon Japan Holland Hills Mori Tower, 18F 5-11.2 Toranomom, Minato-ku Tokyo 105-0001 Japan Phone: +81.3-5733-3320 Fax: +81.3-5733-3321 <a href="mailto:lonsupport@echelon.co.jp">lonsupport@echelon.co.jp</a>

Region	Languages Supported	Contact Information
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 <a href="mailto:lonsupport@echelon.com.cn">lonsupport@echelon.com.cn</a>
Other Regions	English Japanese	Phone: +1.408-938-5200 Fax: +1.408-328-3801 <a href="mailto:lonsupport@echelon.com">lonsupport@echelon.com</a>

# Introduction

This chapter provides an overview of freely programmable modules (FPMs) and explains the types of tasks FPMs can perform. It describes the types of custom embedded applications and drivers you can create with FPMs. It explains how to create and configure FPMs. It summarizes how to create FPM application licenses in order to protect your FPM applications and make them available to customers for order. It provides a quick-start exercise that you can use to create a simple FPM application.

---

## Overview of Freely Programmable Modules

Freely Programmable Modules (FPMs) are custom C/C++ applications and drivers that you can use to customize the functionality of the SmartServer. You can use FPM applications to supplement the built-in applications on the SmartServer and provide solutions for a number of applications, including energy optimization, data analysis, lighting control, and room control. You can use FPM drivers as gateways to legacy systems or nonnative networks such as BACnet and CAN (requires an external interface, sold separately). You can then use your FPM drivers to send data from the SmartServer's RS-232 or RS-485 ports to a built-in application on the SmartServer (for example, a Scheduler or a Data Logger) or a custom SmartServer Web page.

An FPM can perform the following tasks:

- Create data points on the SmartServer.
- Execute code upon data point updates.
- Read and write data to data points.
- Control timers and execute code upon their expiration.
- Read and write data to the RS-232 serial port on the SmartServer.
- Read and write data to the RS-485 serial port on the SmartServer.

---

### *FPM Types*

You can create two types of FPMs: FPM applications and FPM drivers. An FPM application reads and writes values to the data points declared in it, executes code upon data point updates, reads data point properties, and controls timers and executes code upon their expiration. A simple example of an FPM application would be one that reads two data points and adds their values together.

An FPM driver creates data points on the SmartServer (not in the LNS database) and provides values for them by reading and writing to the RS-232 and RS-485 ports on the SmartServer. You can use an FPM driver to create gateways for nonnative devices. An FPM driver can then be used to supply data from the RS-232 or RS-485 ports to a built-in application or custom Web page on the SmartServer.

---

### *Creating and Deploying FPMs*

You can create FPMs using the *i.LON SmartServer 2.0 Programming Tool*, which includes a pre-configured C/C++ Eclipse environment and all the tools needed to write, compile, and upload, your FPMs. After you create your FPMs, you can deploy them on your SmartServer.

Before you can begin writing your FPMs, you need to use the NodeBuilder Resource Editor to create a user-defined functional profile template (UFPT), which defines the set of network variables and configuration properties to which your FPM will read and write.

You can write an unlimited number of FPMs using the demo or full versions of the *i.LON SmartServer 2.0 Programming Tool*. To compile your FPMs and deploy them on your SmartServer, you must use the full version of the *i.LON SmartServer 2.0 Programming Tool*, which is included on the *i.LON SmartServer 2.0 Programming Tools DVD*. To order the *i.LON SmartServer 2.0 Programming Tools DVD* (Echelon part number 72111-409), contact your Echelon sales representative.

To begin running your FPMs on your SmartServer, an FPM programming license must be installed on your SmartServer. To order an FPM programming license for your SmartServer, go to the *i.LON SmartServer 2.0 Web site* at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate). If you plan on using licensed Echelon first-party FPMs or licensed third-party FPMs on your SmartServer, you must also order a separate FPM application license from Echelon or the third-party FPM vendor.

To deploy an FPM application on the SmartServer, you upload the FPM executable module to the SmartServer, and then add an internal device to the SmartServer. The internal device must use a static interface if you are integrating your FPM applications with another LNS application such as the LonMaker tool. If you are running your network with the SmartServer operating as a standalone network manager, the internal device can use a static or dynamic interface.



You can create up to 10 internal FPM devices on the SmartServer. Each internal FPM device may have multiple functional blocks, and the functional blocks may represent different instances of the same FPM application or they may represent instances of different FPM applications. For example, you can create an internal device that has two functional blocks that are both instances of the same FPM application for adding the values of two data points. Alternatively, you can create an internal FPM device that has two functional blocks, where one is an instance of an FPM application that adds data point values, but the other is an instance of an FPM application that subtracts data point values.

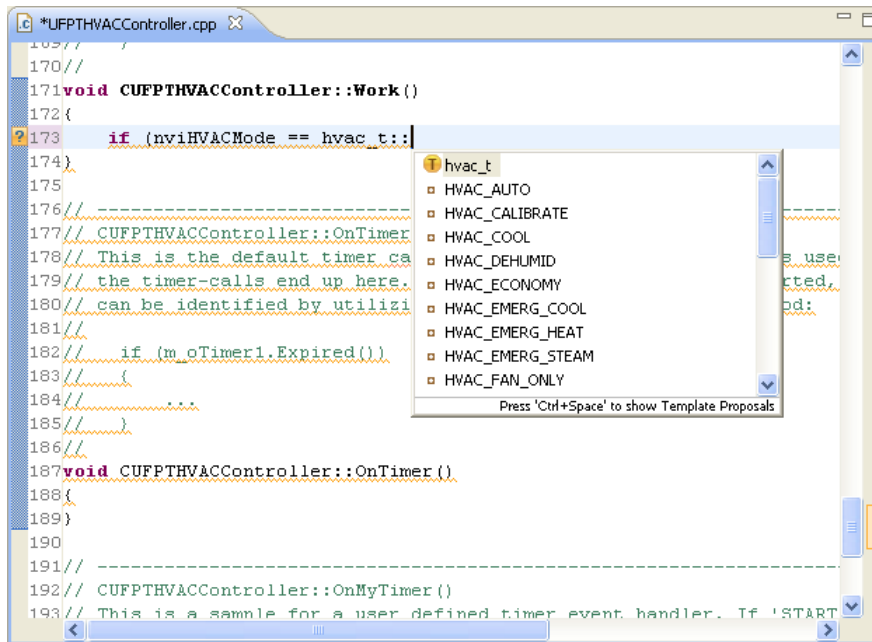
See *Quick-Start FPM Exercise* later in this chapter to create a simple FPM application and deploy it on your SmartServer.

---

## Using Eclipse Environment Commands

The Eclipse environment includes a **Content Assist** command (CTRL + SPACEBAR), an **Open Declaration** command (F3), and ToolTips that you can use when writing your FPMs.

You can use the **Content Assist** command (CTRL + SPACEBAR) when typing elements in the FPM API to open a dialog that lists all the applicable code elements for the snippet you typed. When the dialog opens, you can either continue typing to filter the list, double-click the element to be inserted, or press ESC to close the dialog.



You can use the **Open Declaration** command (F3) to navigate to the declaration of the selected element. This is particularly useful for elements in Echelon's FPM API. For example, you can use this command on the internal data point Write() method, and the method is detailed in the FPM header file that opens.

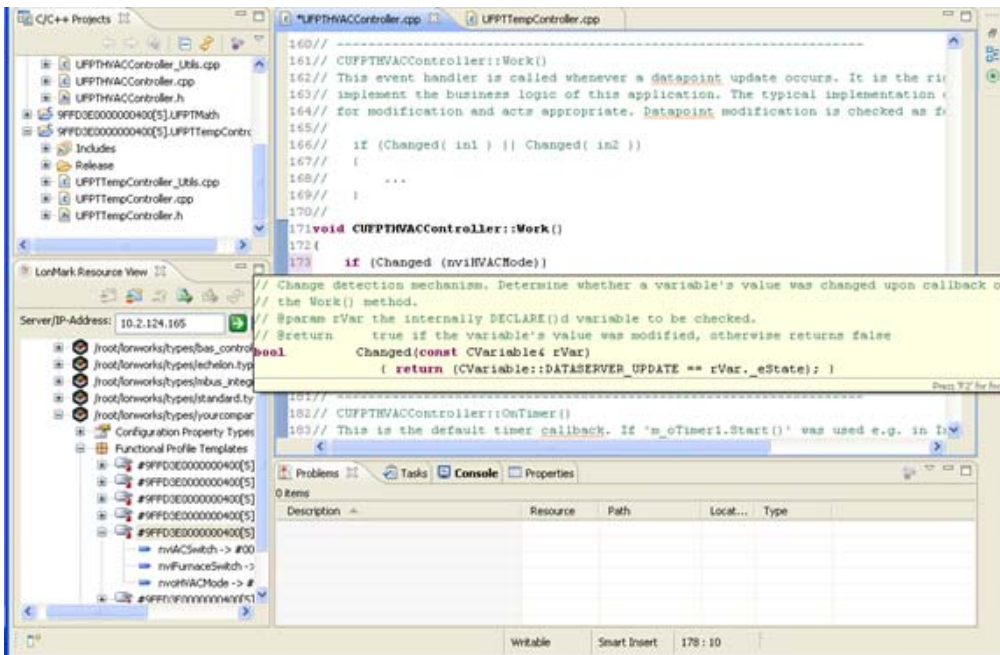
```

*UFPTHVACController.cpp  FPM.h
// Explicitly write the value of an internally DECLARE()d variable. Normally this
// because whenever an internal variable's value was modified, the change will be
// propagated to the dataserver after the Work() or timer routine(s) are finished
// Caution: the implementation of this method is very resource consuming, so use
// @param rVar internally DECLARE()d variable.
// @return status indicating if the execution of the method caused an error (0
// went through (OK)
STATUS Write(CVariable& rVar);
// Reset the priority of an internally DECLARE()d variable.
// Caution: the implementation of this method is very resource consuming, so use
// @param rVar internally DECLARE()d variable.
// @nPrioAuthority the priority of this block-instance used for set() and reset()
// @return status indicating if the execution of the method caused an error
// went through (OK)
STATUS ResetPriority(CVariable& rVar, unsigned short nPrioAuthority);

// =====
// ==> Timer methods
public:
// Convenience method to stop all timers that are currently run by this block-inst
// @return true if all block-instance timers could be stopped, otherwise false.
bool StopAllTimers();

```

You can move your mouse pointer over an object in your FPM to open a tooltip that provides a description and the API of the selected object.



## Debugging FPMs

The SmartServer uses a VxWorks® real-time operating system to run its embedded applications. If you need a source level debugger (VxWorks 6.2 - Wind River Workbench 2.4) or access to VxWorks system calls not encapsulated in the Echelon FPM API, contact Wind River® sales at [www.windriver.com/company/contact/index.html](http://www.windriver.com/company/contact/index.html) for more information on ordering “WindRiver Platform for Industrial Services V3.2 for MIPS32 Processors”.

If you plan on debugging your FPMs with Wind River Workbench, you need to backup and then delete the current **iLonSystem** image on your SmartServer flash disk, copy the **iLonSystemWdb** or **iLonSystemWbdEnd** image in the LonWorks/iLON/Development/Debug/ES\_Debug.<software version> folder on your computer to your SmartServer flash disk, re-name the **iLonSystemWdb** or

**iLonSystemWbdEnd** image on your SmartServer to **iLonSystem**, reboot the SmartServer, create a debug configuration of your FPM in the *i.LON* SmartServer 2.0 Programming Tool and upload it to your SmartServer, and then connect the Workbench debugger to the **iLonSystemWbd** or **iLonSystemWbdEnd** image on your computer via the target server. For more information on how to do this, see Chapter 5, *Creating FPMs*.

If you are not using Wind River Workbench to debug your FPMs, you can still perform some debugging by adhering to the following guidelines when developing your FPMs:

1. Physically connect the computer running the *i.LON* SmartServer 2.0 Programming Tool to the *i.LON* console port using an RS-232 null modem cable. This enables you to use a Terminal emulator such as Windows HyperTerminal to view the *i.LON* console port and debug your FPMs during runtime. After the FPM is initialized you can use Telnet to view the *i.LON* console port.
2. Back up the FPM project frequently. Always make a back up after you make significant changes to an FPM application and successfully compile it.
3. Bracket comments around those portions of the FPM application that you have written. For example, you can do the following:

```
// mycode - begin -----  
out1 = in1 + in2;  
// mycode - end -----
```

4. Add your user help functions to the UFPT<FPM>\_Utils.cpp file (this file is created when you create a new FPM project). This further isolates your code for debugging, and it enables you to port the code over to another FPM project.
5. Insert `printf()` statements in your code frequently. This enables you to do some debugging with the console port of the *i.LON* during runtime, as the console port will receive the `printf()` statements. For example, you can do the following:

```
printf ("[%s %i] value of %s: %d",  
        __FILE__,  
        __LINE__,  
        in1.GetDpPropertyAsString(FPM::Dp::cfgUCPTname),  
        *in1);
```

**Note:** The console port displays the status of your FPM during a reboot.

It is especially important to follow these guidelines because the compiler errors you may receive may only have a generic description that does not indicate which line of code caused the error. In addition, the errors may not appear on the actual line of code causing the error; instead, an error may appear one or two lines above the incorrect code.

---

## *Creating FPM Application Licenses*

You can create FPM application licenses and make your FPMs available to customers, who can implement your third-party FPMs on their SmartServers. To create FPM application licenses, you need to create an FPM licensing tool, enable license validation in your FPM application, build a release version of your licensed FPM, use the *i.LON* License Generator program to create FPM license files, and then provide the FPM licenses and other required files to customers. Customers must order a separate FPM application license from you to use one of your licensed FPMs on their SmartServers.

The *i.LON* SmartServer 2.0 Programming Tools includes an *i.LON* License Generator that you can use to create your own FPM application licensing tool. The *i.LON* License Generator includes the *i.LON* License Generator user interface, a sample security DLL file (**LicenseSecurityHMACMD5.dll**), and a sample XML file (**iLONLicenseGenValuesSample.xml**) that provides the structure and sample pre-defined values of the *i.LON* License Generator user interface. To create your FPM application licensing tool, you need to build a security DLL file named **LicenseSecurity.dll**, and create a license generator configuration file named **iLONLicenseGenValues.xml** that has pre-defined values for the *i.LON* License Generator user interface. Note that if you cannot build the security DLL file, you can

just re-name the sample DLL file **LicenseSecurity.dll**; however, you are solely responsible for the creation of the security DLL file.

You also need to modify your FPM application so that it can check whether a customer's SmartServer has a valid FPM license file for running your FPM. This entails writing a separate license validation routine in your FPM application that (1) checks whether the Lock ID (MACID, LUID, or other user-defined lock type) specified in the FPM license file matches the one on the customer's SmartServer, and (2) checks whether the license key in the FPM license file is valid.

Once you have created the security DLL and license generator configuration files and you have enabled license validation in your FPM application, you can use the *i.LON* License Generator user interface to create FPM application licenses. The FPM application license is an XML file that must be on a SmartServer in order for it to run your licensed FPM. This means that you can make your FPMs available to customers, while protecting your FPMs from unauthorized use or piracy.

A customer who orders a licensed FPM application from you must install the FPM application license for that FPM on their SmartServer. If a customer attempts to run your FPM application without installing the FPM license file on their SmartServer, the SmartServer will report a license error and the FPM will not run on the SmartServer. This means that a customer running one of your FPMs must have two licenses installed on their SmartServer: an FPM programming license from Echelon and the FPM application license from your company that you generated for that FPM.

See Chapter 7, *Creating FPM Application Licenses*, for more information on creating FPM licenses and enabling license validation in your FPM application.

---

## Quick-Start FPM Exercise

The following section describes how to create a simple FPM application called UFPTMath and then deploy, test, and connect the FPM on your SmartServer. The Math application adds two **SNVT\_count** input data points and stores the result in a **SNVT\_count** output data point. To create, deploy, test, and connect the Math FPM application, you perform the following steps:

1. With the NodeBuilder Resource Editor, create a user-defined functional profile template (UFPT) for the FPM, and then generate your company's FPM resource file set. Copy your company's updated FPM resource file set to the root/IonWorks/types/User/<YourCompany> folder on the SmartServer flash disk.
2. With a text editor such as Notepad, create a model file (.nc extension) in which you declare all the data points in the UFPT, and a functional block that implements an instance of the UFPT. With the *i.LON* SmartServer 2.0 LonWorks Interface Developer tool, generate a device interface (XIF) file from your model file. Copy the XIF to the root/IonWorks/Import/<YourCompany> folder on the SmartServer flash disk.
3. With the *i.LON* SmartServer 2.0 Programming Tool, create a new FPM project from the UFPT you created in step 1.
4. With the *i.LON* SmartServer 2.0 Programming Tool, write the FPM application and then build it.  
**Note:** The full version of the *i.LON* SmartServer 2.0 Programming Tools must be installed on your computer to build the FPM.
5. On your SmartServer, deploy the FPM. To do this you upload the FPM application to the root/modules/User/<YourCompany> folder on the SmartServer flash disk. You then add a new internal device to the SmartServer that uses the device interface (XIF) file that you created for the FPM.
6. On your SmartServer, test the FPM. To do this, you open the **View – Data Points** Web page, add the input and output data points in the FPM application, update one of the input data points, and observe that the output data point is updated accordingly.

**Note:** An FPM programming license must be installed on your SmartServer for the FPM to run on your SmartServer. You can order a FPM programming license from the *i.LON SmartServer 2.0* Web site at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate)

7. On your SmartServer, create Web connection between the data points declared in your FPM application and the data points on the SmartServer, and then use the **View – Data Points** Web page to test that the Web connections are updating the FPM data points.

**Tip:** Review the guidelines for creating FPMs that are listed in the *Debugging FPMs* section. You should follow these guidelines in order to help debug your FPM applications.

---

## Step 1: Creating and Copying the FPM Template

You can create the template to be used by the FPM. To create a new template, you create a new user-defined functional profile template (UFPT) in the NodeBuilder Resource Editor (the NodeBuilder Resource Editor is included with the *i.LON SmartServer 2.0* software). After you create a new UFPT, you select the standard and user-defined network variable and configuration property types (SNVTs, UNVTs, SCPTs, and UCPTs) to which the FPM will read and write. Once you have added all the data types to be use by the FPM, you generate the updated FPM resource file set in which the template was created and copy your resource files to the root/lonWorks/types/User/<YourCompany> folder on the SmartServer flash disk

To create the FPM template, and generate and copy the updated FPM resource file set, follow these steps:

1. Start the NodeBuilder Resource Editor. To do this, click **Start**, point to **Programs**, point to **Echelon NodeBuilder Resources**, and then click **NodeBuilder Resource Editor**.

**Note:** If NodeBuilder Resource Editor is not installed on your computer, you can install NodeBuilder Resource Editor 3.14 from the *i.LON SmartServer 2.0* DVD or the *i.LON SmartServer 2.0 Programming Tools* DVD. See the *i.LON SmartServer 2.0 User's Guide* for more information on installing the NodeBuilder Resource Editor.

2. Create a new resource file set for your company. If you plan on integrating your FPM applications with an LNS application such as the LonMaker tool, you need to create a new scope 5 resource file set. To create a new resource file set, follow these steps:

- a. Right-click your company's resource file set and click **New Resource File Set** on the shortcut menu. The **New Resource File Set** dialog opens

**Note:** If your company does not already have a folder under the LonWorks/types/User directory on your computer, you need to create one. This folder will be used to store the resource file set you will create for your FPMs. To create a new folder, right-click the **LonWorks/types/Ldrf.Cat** file and then click **Add Folder** on the shortcut menu.

- b. If you plan on integrating your FPM applications with an LNS application such as the LonMaker tool, you should select **5** in the **Scope** box (this sets the scope to device class, manufacturer, usage, and channel type).
- c. In the manufacturer (**MMMMM**) field of the **Program ID** box, enter your 5-digit manufacturer ID in hexadecimal format.

**Note:** If your company does not have a manufacturer ID, you can get a temporary manufacturer ID from LonMark at [www.lonmark.org/mid](http://www.lonmark.org/mid). In addition, if your company has many FPM developers, it is recommended that you request temporary manufacturer IDs for them. After you obtain your temporary manufacturer ID, you can enter it in the **MMMMM** field of the **Program ID** box.

- d. In the format (**F**) field of the **Program ID** box, enter 9 (this sets the Standard Development Program ID flag).

- e. In the channel (**TT**) field of the **Program ID** box, enter **04** if you have an FT-10 model of the SmartServer or enter **10** if you have a PL-20 model of the SmartServer.
  - f. In the **Resource File Set Name** box, enter “FPM Development”, “FPM Examples”, or something comparable
  - g. Click **OK**.
3. Expand the folder containing your company’s FPM resource file set, right-click the **Functional Profile Template** folder, and then click **New FPT** on the shortcut menu.
  4. Enter **UFPTMath** for the name of the new UFPT you created.
  5. Double-click **UFPTMath**, or right-click it and click **Open** on the shortcut menu. The **Modify Functional Profile** dialog opens.
  6. Expand the **LonWorks/types/STANDARD** directory, expand the **Network Variables** folder, and then click and drag the **SNVT\_count** data point to the **Mandatory NVs** folder. In the **Name** property, enter **in1**.
  7. Repeat step 6, but name the new data point **in2**.
  8. Repeat step 6, but name the data point **out1** and select **Output** under the **NV Settings** box.
  9. Click **OK**.
  10. Generate your company’s updated FPM resource file set. To do this, right-click your company’s FPM resource file set, and then click **Generate Resources Set** on the shortcut menu. The **Generate Resources Set** dialog opens.
  11. Click **Yes** to generate the updated FPM resource file set.
  12. Copy your company’s updated FPM resource file set from the LonWorks\Types\User\*<Your Company>* folder on your computer to the root/LonWorks/Types/User/*<Your Company>* folder on the SmartServer flash disk. Note that you may need to create your User/*<YourCompany>* folder on the SmartServer flash disk before copying your resource file set.

For more information on creating UFPTs for your FPMs, see Chapter 3, *Creating FPM Templates*.  
 For more information on using the NodeBuilder Resource Editor, see the *NodeBuilder Resource Editor User’s Guide*.

---

## ***Step 2: Creating and Copying the Device Interface (XIF) File***

You can create the device interface used by your FPM. To create the device interface, you use a text or programming editor such as Notepad to write a model file (.nc extension). In this model file, you declare all the network variables and configuration properties that you added to the UFPT, and you declare a functional block that implements an instance of that UFPT. After you create the model file, you open a Command Prompt window and use the *i.LON* SmartServer 2.0 LonWorks Interface Developer tool to convert your model file to a device interface (XIF) file. You then copy the XIF (.xif extension) to the root/lonWorks/Import/*<YourCompany>* folder on the SmartServer flash disk.

To create the device interface (XIF) file your FPM application, follow these steps:

1. Open a text or programming editor on your computer such as Notepad.
2. Enter or copy the following code, which does the following: declares the three **SNVT\_count** network variables in **UFPTMath**, declares a functional block that instantiates **UFPTMath**, lists implementations of the three **SNVT\_count** network variables in **UFPTMath**, and defines a name and an external name for the functional block.

```
network input SNVT_count in1;
network input SNVT_count in2;
network output SNVT_count out1;
```

```

fblock UFPTMath {
  in1 implements in1;
  in2 implements in2;
  out1 implements out1;
} fbMathFunction external_name ("Math Function");

```

3. Save your model file on your computer. This example stores a model file named “math.nc” in a folder named “ModelFile” that has been created under the C:\LonWorks directory. The file path of the source file in this example is therefore C:\LonWorks\ModelFile\math.nc.
4. Create a <YourCompany> folder for your company under the C:\LonWorks\Import folder if one does not already exist. This is where the XIF generated by the *i.LON SmartServer 2.0 LonWorks Interface Developer* tool is to be stored.
5. Install the full version of the *i.LON SmartServer 2.0 Programming Tools* from the *i.LON SmartServer 2.0 Programming Tools DVD* if it is not already installed on your computer. This will install the *i.LON SmartServer 2.0 LONWORKS Interface Developer* tool. You will use this command line interface in the next step to generate the device interface (XIF) file used by your FPM. For more information on installing the *i.LON SmartServer 2.0 LonWorks Interface Developer* tool, see Chapter 2, *Installing i.LON SmartServer 2.0 Programming Tools*.
6. Convert your model file to a XIF using the *i.LON SmartServer 2.0 LonWorks Interface Developer* tool. To do this, open a Command Prompt window and then type the following command:

```

libilon --source=<model file path> --pid=<program ID> --
out=<destination path> --basename=<XIF name >

```

For this example, you would type the following at the command prompt (you need to replace the sample program ID with your company’s program ID, and you need to replace the “YourCompany” folder in the C:\LonWorks\Import directory with your company’s folder):

```

libilon --source=C:\LonWorks\ModelFile\Math.nc --
pid=9F:FD:3E:00:00:00:04:00 --out=C:\LonWorks\Import\YourCompany
--basename=Math

```

This creates device interface files named “Math” (.xif and .xfb extensions), and stores them in the C:\LonWorks\Import\<YourCompany> folder.

**Note:** You need to separate the command switches (--source, --pid, --out, and --basename) with spaces, but you do not insert spaces between the command switch and the specified argument.

7. Copy the XIF (.xif extension) generated in step 5 to the root/lonWorks/Import<YourCompany> folder on the SmartServer flash disk. Note that you may need to create the <YourCompany> folder on the SmartServer flash disk before copying the XIF.

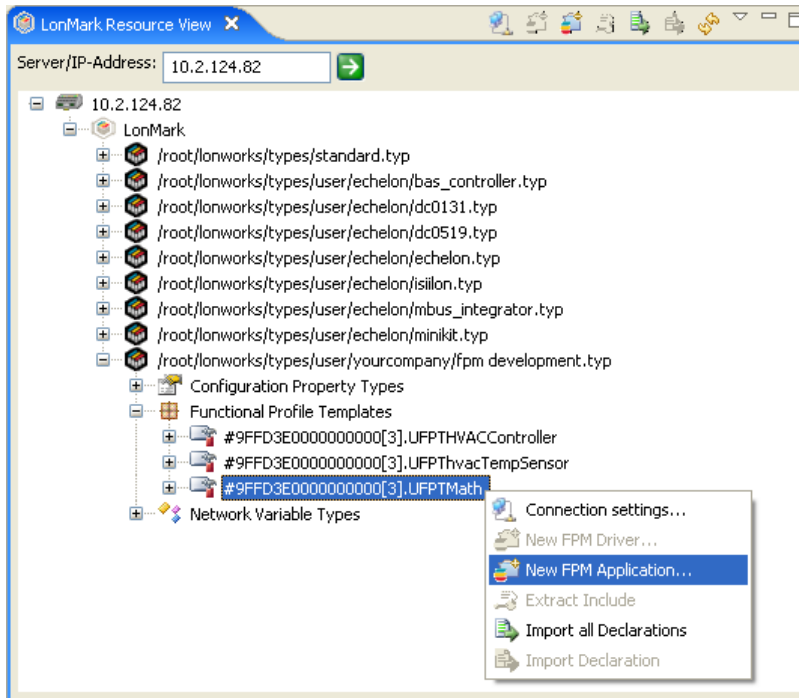
For more information on creating XIFs for your FPMs, see Chapter 4, *Creating FPM Device Interface (XIF) Files*.

### Step 3: Creating the FPM Project

You can create the FPM project using the *i.LON SmartServer 2.0 Programming Tool*. To create a new FPM project, you make your SmartServer accessible to the *i.LON SmartServer 2.0 Programming Tool*, and then create a new FPM project from the resource file set you added to the SmartServer flash disk. To create an FPM application, follow these steps:

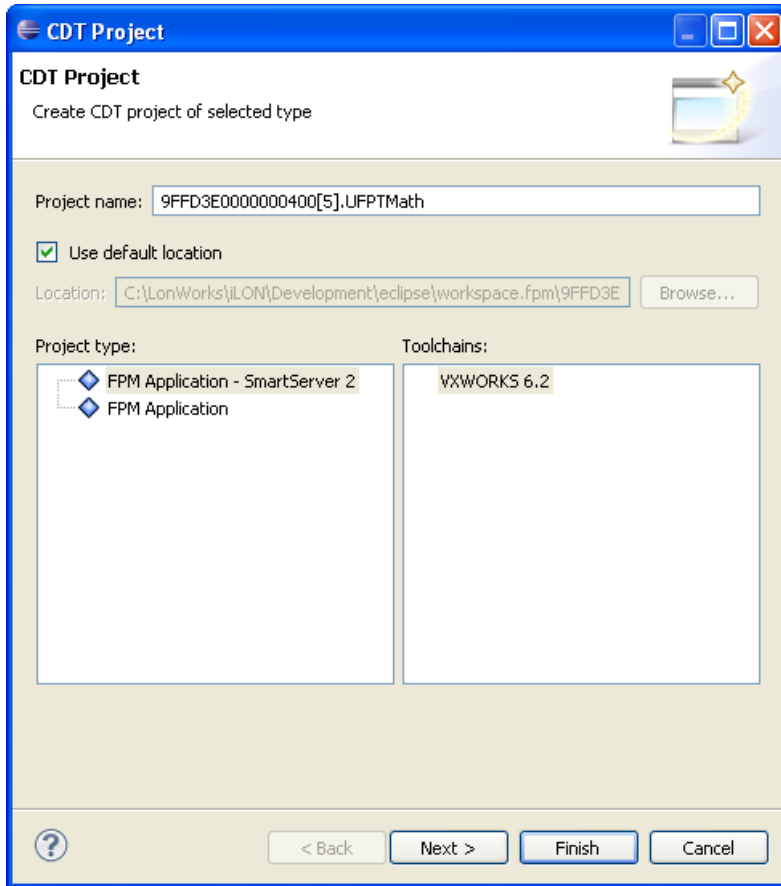
1. Start the *i.LON SmartServer 2.0 Programming Tool*. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0 Programming Tool* opens.
2. Locate the **LonMark Resource View** at the bottom left-hand corner of the document window.

3. In the **Server/IP-Address** box in the **LonMark Resource View**, enter the hostname or IP address of your SmartServer and then click the Go button to the right.
4. Expand the SmartServer icon and then expand the LonMark folder. The resource files in the root/lonWorks/types folder on your SmartServer flash disk are shown.
5. Expand your company's FPM resource file set, expand the **Functional Profile Templates** folder, right-click the *<company program ID>.UFPTMath* template, and then click **New FPM Application** on the shortcut menu.



6. The **CDT Project** dialog opens.

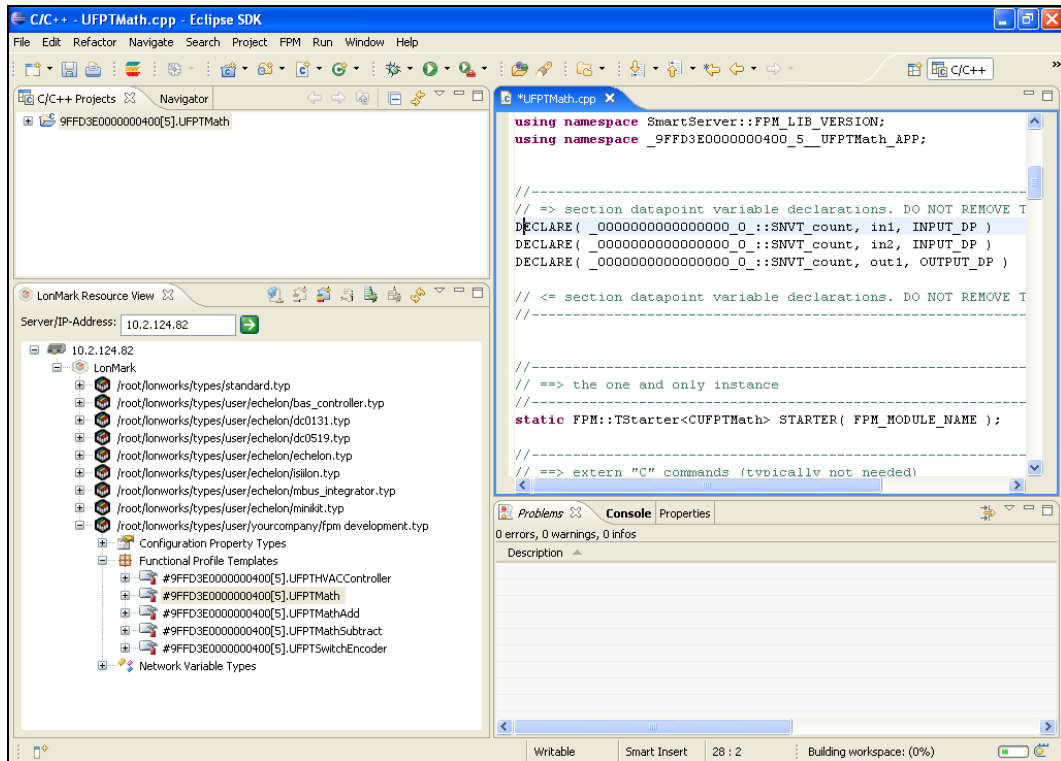




7. If you will be deploying this example FPM on a SmartServer 2.0, click **Finish** to accept the default settings. If you will be deploying this example FPM on a SmartServer 1.0, click **FPM Application**, and then click **Finish**.

**Note:** Avoid modifying the default project name. This ensures that the your FPM project has a unique namespace by following the FPM naming convention, which is *<company program ID>.UFPT< FPT Name>*.

8. A new UFPTMath FPM project folder with the name specified in step 4 is added to the **C/C++ Projects** view, and the source file view opens to the right of the **C/C++ Projects** view.



9. You can observe that the Data Point Variable Declarations section automatically includes DECLARE statements for each data point in the .UFPTMath template.

For more information on creating FPM projects, see Chapter 5, *Creating FPMs*.

## Step 4: Writing the FPM Application

You can write the FPM application using the *i*.LON SmartServer 2.0 Programming Tool. This mainly entails specifying the logic to be executed on the data points declared in the application. For this exercise,<sup>75</sup>

you create a simple addition algorithm in the `Work()` routine of the `UFPTMath.cpp` file. Whenever one of the input data points is updated, the algorithm adds the two input data points and stores the result in the output data point. Note that the `Work()` routine is one of four routines in the FPM application. The other three routines are `Initialize()`, `OnTimer()`, and `Shutdown()`.

- The `Initialize()` routine is executed when the FPM application is started or enabled. You can initialize data point values and start timers in this routine.
- The `Work()` routine is executed when data points declared in the FPM application are updated. You can write values to the data points declared in your FPM application and read data point properties in this routine.
- The `OnTimer()` routine is executed when a timer created in the `Initialize()` routine expires. You can read data point properties in this routine.
- The `Shutdown()` routine is executed when the FPM application is stopped or disabled. You can stop timers and perform any required cleanup in this routine.

To write and build the FPM application, follow these steps:

1. In the `UFPTMath.cpp` file, find the `Work()` routine.
2. In the `Work()` routine, enter the following code:

```

void CUFPTMath::Work()
{
    if (Changed(in1) || Changed(in2))
    {
        out1 = in1 + in2;
        printf("%i + %i = %i \n", *in1, *in2, *out1);
    }
}

```

3. Build the FPM application. To do this, click **File** and then click **Save**. The FPM executable module (.app extension) is updated. If the build is not performed, click **Project** and then click **Build Project**. You can then click **Project** and select **Build Automatically** so that your FPM applications are built automatically when you save them.

**Note:** If a dialog appears prompting you to enter a license, you need to install the full version of the *i.LON SmartServer 2.0 Programming Tools* on your computer in order to build your FPM application. To order the full version of the *i.LON SmartServer 2.0 Programming Tools*, contact your Echelon sales representative.

For more information on writing FPM applications and FPM drivers, see Chapter 5, *Creating FPMs*.

---

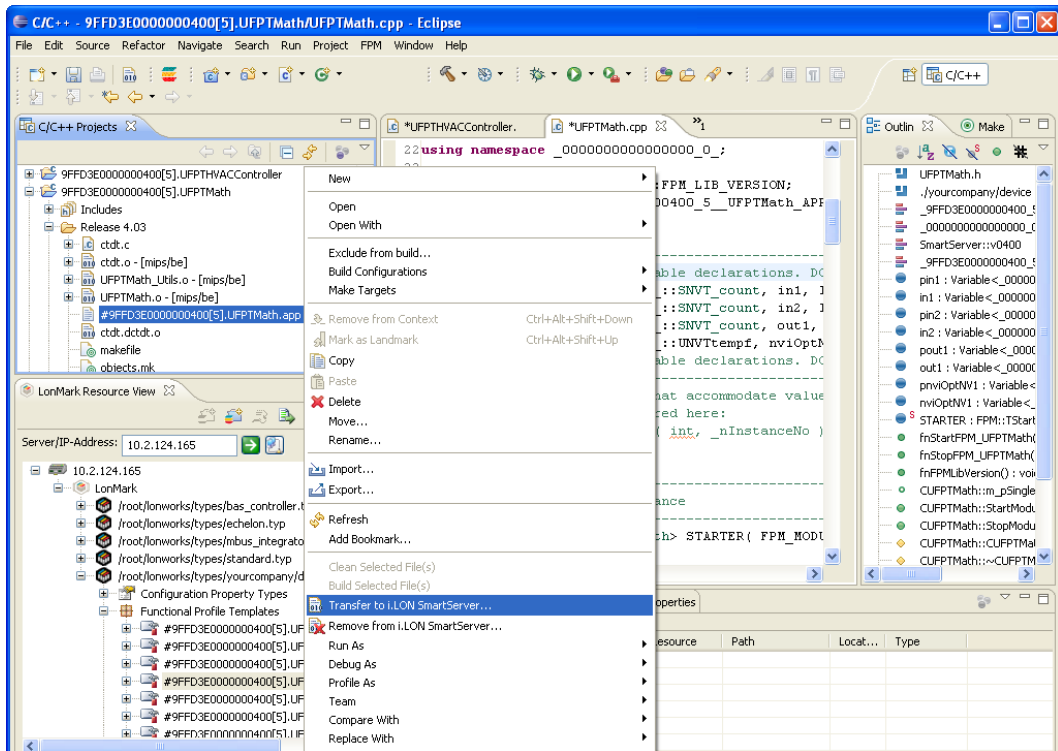
## ***Step 5: Deploying the FPM Application on a SmartServer***

You can deploy your FPM application on your SmartServer. To do this you upload your FPM application to the SmartServer flash disk and then add a new internal device on the SmartServer tree that uses the device interface (XIF) file that you created for the FPM in *Step 2: Creating and Copying the Device Interface (XIF) File*.

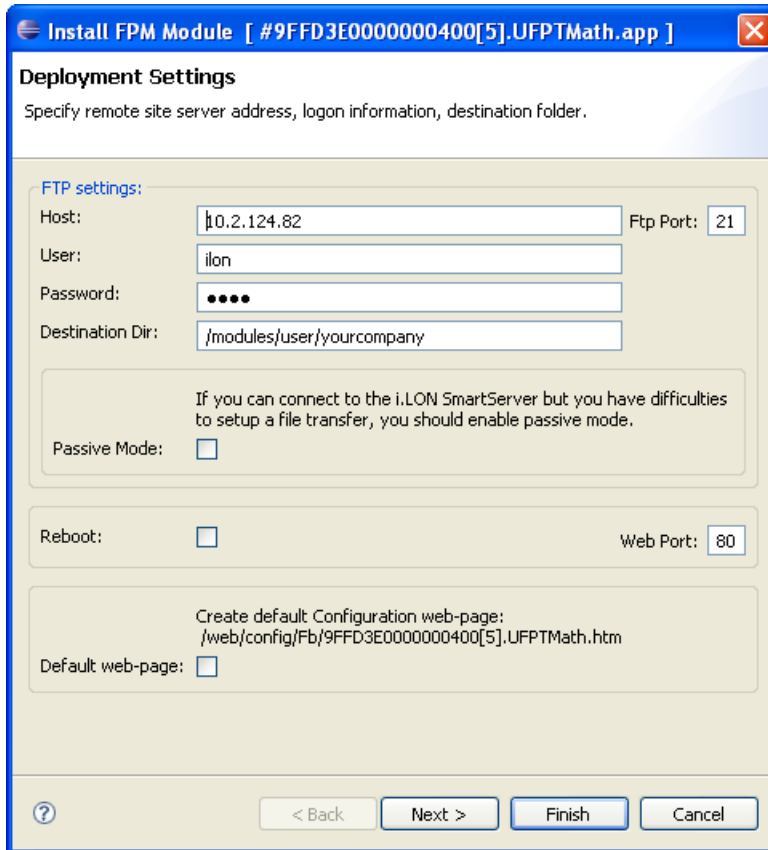
### *Uploading the FPM Application*

To upload the FPM application to your SmartServer, follow these steps:

1. Create a User/<YourCompany> folder under the root/modules folder on the SmartServer flash disk if one does not already exist. This is where the executable module generated by the *i.LON SmartServer 2.0 Programming Tool* should be stored.
2. In the **C/C++ Projects** view, expand the **Release 4.03** folder, right-click the <company program ID>.UFPTMath.app file, and then click **Transfer to i.LON SmartServer** in the shortcut menu.



3. The **Install FPM Module** dialog opens with the Deployment Settings window.



4. Enter the following information if different than the defaults:
  - The IP address or hostname of the SmartServer to which the FPM is to be uploaded. The default is the SmartServer IP address or hostname entered in the LonMark Resource View.
  - The user name and password used to access the SmartServer FTP server. The default user name and password are both “ilon”.
  - The directory on your SmartServer flash disk to which your FPM application is to be stored. The default directory is root/modules. You should create a root/modules/User/<YourCompany> folder on the SmartServer flash disk and store your FPM application in that folder.
  - The port used to access the FTP server on your SmartServer. The default port is **21**.
  - The port used by your SmartServer to transmit and receive SOAP/HTTP requests. The default port is **80**, but you may change it to any valid port number. Contact your IS department to ensure your firewall is configured to allow access on this port.

See Chapter 6, *Deploying Freely Programmable Modules*, for more information on the settings in this dialog.

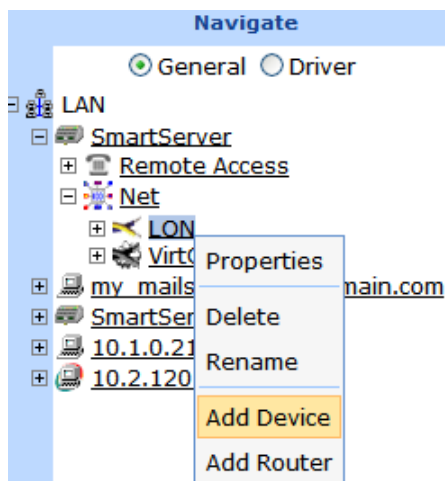
5. Click **Finish**. The FPM application is uploaded to your SmartServer. You can use the console port to verify that the FPM is being uploaded to your SmartServer. Once the FPM application has been uploaded, you can proceed to the next step, which is creating an internal FPM device.

**Note:** If FPM Programming is not licensed on your SmartServer, the console port will display messages stating that the FPM license is invalid, the FPM feature is not properly licensed, and FPM tasks cannot be created. You can order a FPM programming license from the *i.LON* SmartServer 2.0 Web site at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate).

### *Creating an Internal FPM device*

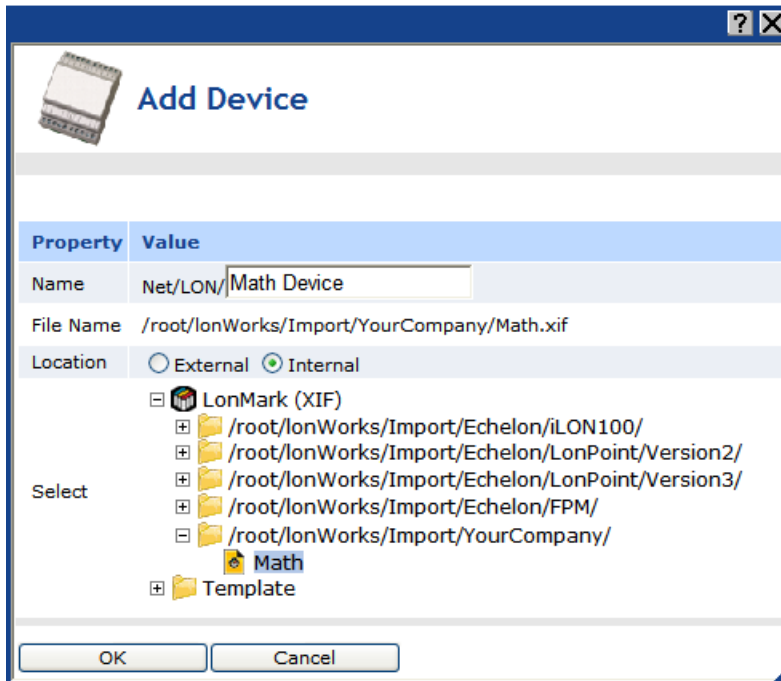
After you upload your FPM application to the SmartServer, you can create an internal FPM device on your SmartServer following these steps:

1. Expand the **Net** network, right-click the **LON** channel, and then select **Add Device** on the shortcut menu.

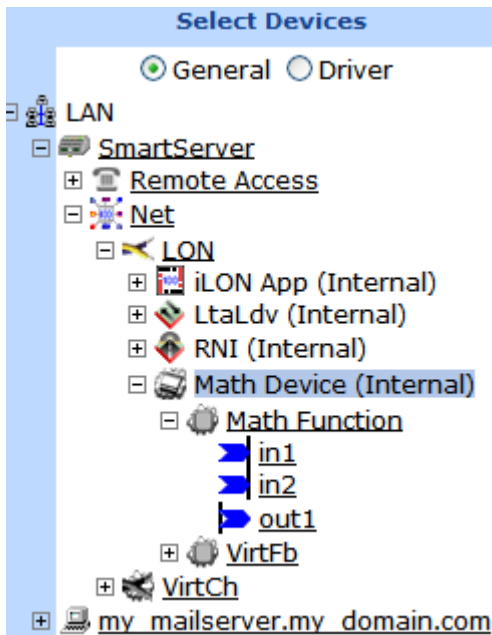


2. The **Create Device** dialog opens.
3. Enter a meaningful name for the device such as “Math Device”.
4. In the **Location** property, select **Internal**.

- Expand the **LonMark (XIF)** folder, expand the **root/lonworks/import/<YourCompany>** folder, and then select the XIF for your Math FPM.



- Click **OK**. A **Math Device (Internal)** device is added to the bottom of the **LON** channel tree.
- Click **Submit**. You must wait for the SmartServer to instantiate the XIF file used for the internal Math device. The time it takes depends on the size of the XIF. Once the XIF has been instantiated, you can expand the **Math Device (Internal)** device and the **Math Function** functional block to show the data points in the FPM application.



---

## Step 6: Testing the FPM Application

After you deploy an FPM application, you can test your FPM application using the **View – Data Points** Web page. To do this, you open the **View – Data Points** Web page, add the input and output data points in the FPM application, update one of the input data points, and observe that the output data point is updated accordingly.

To test an FPM application on your SmartServer, follow these steps:

1. Click **View** and then click **Data Points**. The **View – Data Points** Web page opens.
2. Close the graph by clicking the ‘X’ in the upper right-hand corner of the application frame.
3. Under the **Math Function** functional block, click the **in1**, **in2**, and **out1** data points. The data points appear in the **View – Data Points** Web page.

The screenshot shows the 'View - Data Points' interface. On the left is a tree view under 'Select Data Point' with 'General' selected. The tree shows a hierarchy: LAN > SmartServer > Remote Access > Net > LON > iLON App (Internal) > LtaLdv (Internal) > RNI (Internal) > Math Device (Internal) > Math Function > in1, in2, out1, VirtFb. On the right is a table with columns: Name, Format, Value, Unit, Priority, Status. A 'Show Graph' link is above the table.

Name	Format	Value	Unit	Priority	Status
0 Net/LON/Math Device/Math Function/in1	SNVT_count	0	units	255	ONLINE
1 Net/LON/Math Device/Math Function/in2	SNVT_count	0	units	255	ONLINE
2 Net/LON/Math Device/Math Function/out1	SNVT_count	0	units	255	ONLINE

4. Enter a different value for either or both of the **in1** or **in2** data points. Observe that **out1** data point is updated and displays the sum of the **in1** or **in2** data points.

This screenshot shows the same 'View - Data Points' interface as the previous one, but with updated values in the table. The 'Value' column now shows 2 for in1, 3 for in2, and 5 for out1.

Name	Format	Value	Unit	Priority	Status
0 Net/LON/Math Device/Math Function/in1	SNVT_count	2	units	255	ONLINE
1 Net/LON/Math Device/Math Function/in2	SNVT_count	3	units	255	ONLINE
2 Net/LON/Math Device/Math Function/out1	SNVT_count	5	units	255	ONLINE

For more information on testing FPMs, see Chapter 6, *Deploying FPMs on a SmartServer*.

---

## Step 7: Connecting the FPM Data Points

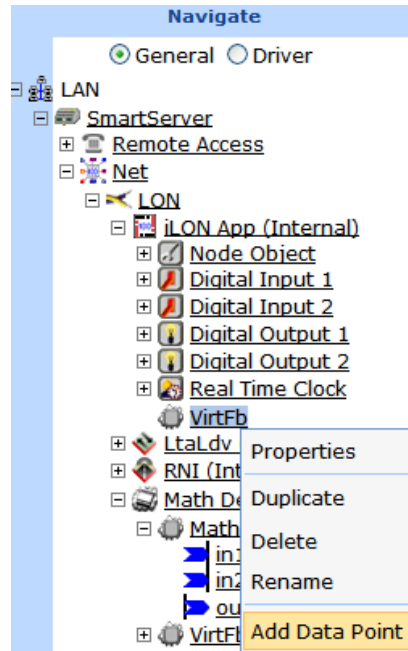
After you verify that your FPM application is functioning properly, you can use Web connections or LONWORKS connections to connect the data points declared in your FPM device to the data points on the SmartServer or to the data point of external devices. Note that the SmartServer must be operating in LNS mode (**LNS Auto** or **LNS Manual**) in order to create LONWORKS connections.

For this quick-start exercise, you will use Web connections to connect the data points in your FPM application to data points on the SmartServer. To do this, you create three dynamic **SNVT\_count** data points, and you create Web connections between the data points declared in your FPM device and the dynamic data points you created on the SmartServer. You can then use the **View – Data Points** Web page to test that changes made to the dynamic data points on the SmartServer are updating the **in1** and

**in2** data points declared in the FPM device and that the **out1** data point in the FPM device is updating the dynamic data point on the SmartServer.

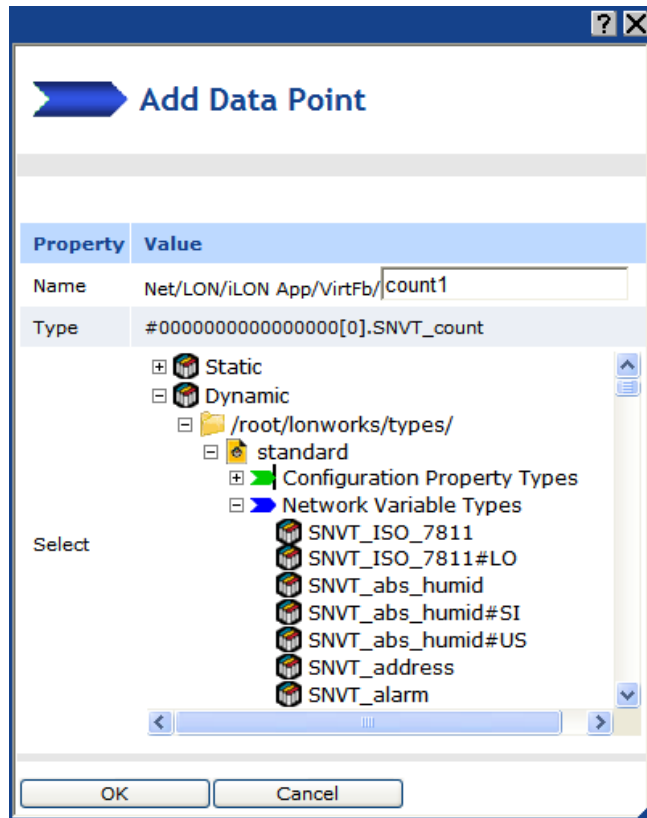
To connect the data points declared in your FPM to the data points on the SmartServer, follow these steps:

1. Create a dynamic **SNVT\_count** data point on the **VirtFB** functional block under the **i.LON App (Internal)** device. To do this, follow these steps:
  1. From the tree, expand the **Net** network, expand the **LON** channel, and then expand the **i.LON App (Internal)** device to show the **VirtFB** functional block at the bottom of its tree.
  2. Right-click the **VirtFB** functional block and then click **Add Data Point** on the shortcut menu.

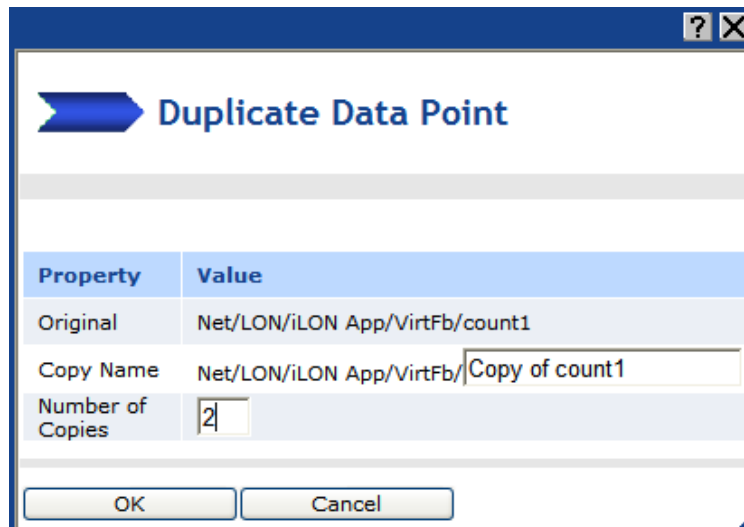


3. The **Add Data Point** dialog opens.
4. In the **Name** property, enter a meaningful name such as “count1”.
5. In the **Select** property, expand the **Dynamic** icon, expand the **root/lonworks/types** directory, expand the **standard** resource files folder, expand **Network Variable Types**, and then click the **SNVT\_count** data point.

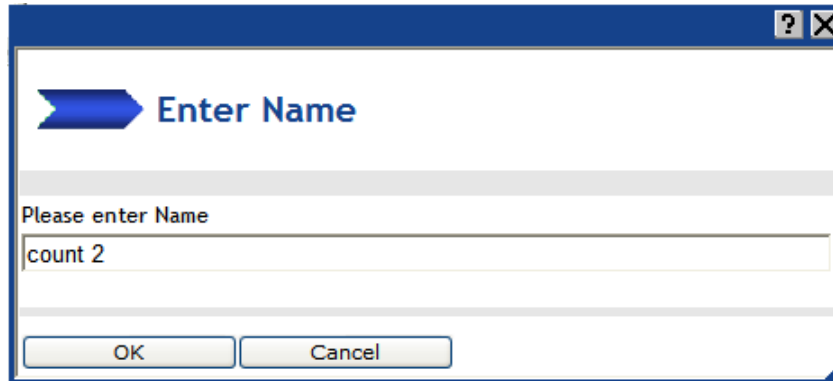




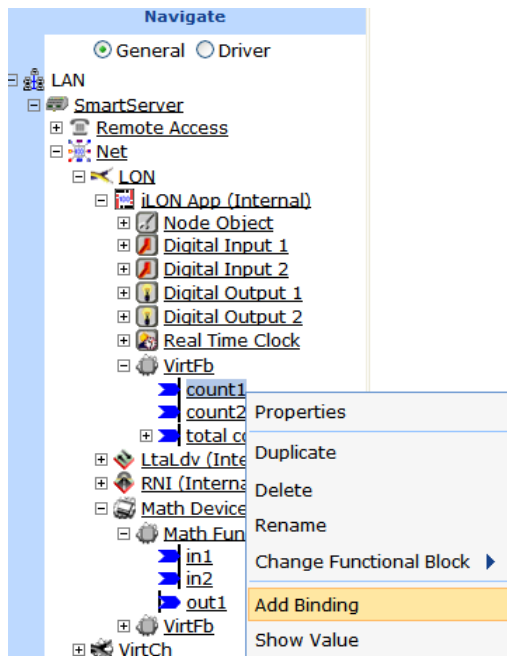
6. Click **OK**. A data point with the name you specified in step d is added underneath the **VirtFB** functional block.
7. Click **Submit**.
2. Create two copies of the dynamic **SNVT\_count** data point that you created in step 1.
  - a. Right-click the dynamic **SNVT\_count** data point, and then click **Duplicate Data Point** on the shortcut menu.
  - b. The **Duplicate Data Point** dialog opens.
  - c. In the **Number of Copies** property, enter **2**.



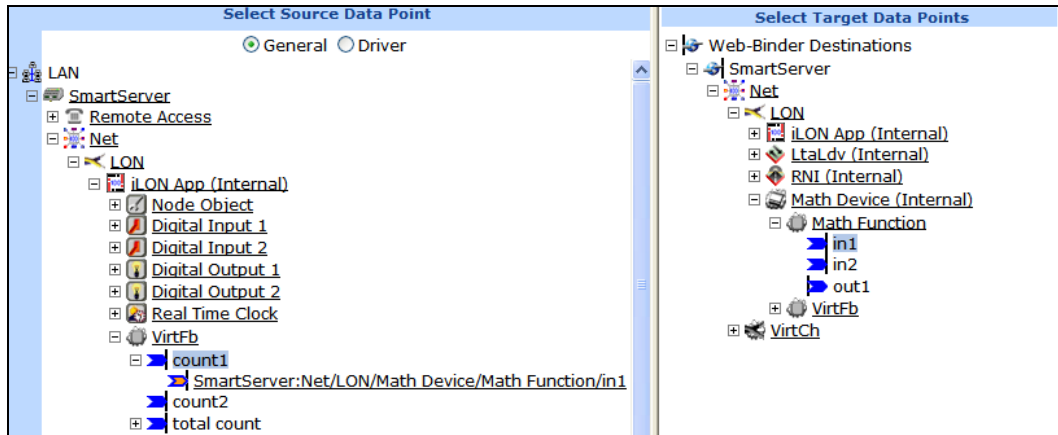
- d. Click **OK**. Two copies of the data point named “Copy of <DP name>” are added underneath the **VirtFB** functional block. Note that the second data point copy will have an index appended to its name.
  - e. Click **Submit**.
3. Re-name the two data point copies you created in step 2. To do this, follow these steps:
    - a. Right-click the first data point copy, and then click **Rename** on the shortcut menu.
    - b. The **Enter Name** dialog opens.
    - c. Enter a meaningful name for the first data point copy such as “count 2”.



- d. Click **OK**. The data point is re-named to the name specified in step c.
  - e. Click **Submit**.
  - f. Repeat steps a–e for the second data point copy. In step c, enter a meaningful name such as “total count”.
4. From the tree in the sidebar (left) frame, right-click the “Net/LON/i.LON App (Internal)/VirtFB/count1” data point and then click **Add Binding** in the shortcut menu.



- The **Configure – Web Binder** Web page opens and the hostnames of the local SmartServer and any remote SmartServers added to the LAN, which are collectively referred to as *Webbinder Destinations*, appear in the application frame to the right.
- From the Webbinder Destinations tree on the right frame, expand the *i.LON* Webbinder Destination icon, expand the **Net** network, expand the **LON** channel, and then expand the **Math Device (Internal)** device, and expand the **Math Function** functional block to show its “in1”, “in2”, and “out1” data points. Click the “Net/LON/Math Device (Internal)/Math Function/in1” data point to specify it as the target data point. A reference to the target “in1” data point is added underneath the “count1” source data point in the tree in the left frame.



- Click **Submit**.
- Following steps 4–7, create a Web connection between the “VirtFB/count2” data point in the tree in the left frame (the source data point) and the “Math Function /in2” data point in the Webbinder Destinations tree in the right frame (the target data point).
- Following steps 4–7, create a Web connection between the “Math Function/out1” data point in the tree in the left frame (the source data point) and the “VirtFB/total count” data point in the Webbinder Destinations tree in the right frame (the target data point).

- Click **View** and then click **Data Points**. The **View – Data Points** Web page opens.
- Close the graph by clicking the ‘X’ in the upper right-hand corner of the application frame.
- Under the Net/LON/ Math Device (Internal)/VirtFB functional block, click the **count1**, **count2**, and **total count** data points to add them to the Web page.

If the **in1**, **in2**, and **out1** data points under the Net/LON/ Math Controller (Internal)/Math Function functional block no longer appear in the Web page, click each of these data points to add them back to the Web page.

- Observe that the **Math Function/out1** data point has the same value as the **VirtFB/total count** data point. The Web connection keeps these two data points synchronized.

View - Data Points						
<a href="#">Show Graph</a>						
	Name	Format	Value	Unit	Priority	Status
0	Net/LON/Math Device/Math Function/in1	SNVT_count	2	units	255	ONLINE
1	Net/LON/Math Device/Math Function/in2	SNVT_count	3	units	255	ONLINE
2	Net/LON/Math Device/Math Function/out1	SNVT_count	5	units	255	ONLINE
3	Net/LON/iLON App/VirtFb/count1	SNVT_count	0	units	255	NUL
4	Net/LON/iLON App/VirtFb/count2	SNVT_count	0	units	255	NUL
5	Net/LON/iLON App/VirtFb/total count	SNVT_count	5	units	255	ONLINE

14. Enter values for both the **VirtFB/count1** and **VirtFB/count2** points. Observe the following:

- The **Math Function/in1** and **Math Function/in2** data points in the FPM are updated to the same values as the dynamic data points on the SmartServer to which they are connected. The Web connections keep these sets of data points synchronized.
- The **Math Function/out1** FPM output data point is updated to reflect the sum of the **in1** and **in2** FPM input points.
- The **VirtFB/total count** data point on the SmartServer is updated to match the new sum stored in the **Math Function/out1** FPM data point.

View - Data Points						
<a href="#">Show Graph</a>						
	Name	Format	Value	Unit	Priority	Status
0	Net/LON/Math Device/Math Function/in1	SNVT_count	3	units	255	ONLINE
1	Net/LON/Math Device/Math Function/in2	SNVT_count	4	units	255	ONLINE
2	Net/LON/Math Device/Math Function/out1	SNVT_count	7	units	255	ONLINE
3	Net/LON/iLON App/VirtFb/count1	SNVT_count	3	units	255	ONLINE
4	Net/LON/iLON App/VirtFb/count2	SNVT_count	4	units	255	ONLINE
5	Net/LON/iLON App/VirtFb/total count	SNVT_count	7	units	255	ONLINE

For more information on using LONWORKS connections and Web connections to connect the data points declared in an FPM device, see Chapter 6, *Deploying FPMs on a SmartServer*. For more information on using the Web Binding application, including how to validate, delete, and add attachments to bindings, see Chapter 4 of the *i.LON SmartServer 2.0 User's Guide*.

# Installing *i*.LON SmartServer 2.0 Programming Tools

This chapter describes how to install, upgrade, and uninstall the *i*.LON SmartServer 2.0 Programming Tools.

---

## Installation and Upgrading Overview

The *i.LON SmartServer 2.0* DVD includes a demo version of the *i.LON SmartServer 2.0* Programming Tools, which you can use to write FPM applications and drivers. You cannot use the demo version, however, to compile and deploy the FPMs. To compile and deploy FPMs, you must use the full version of the *i.LON SmartServer 2.0* Programming Tools. The full version of the *i.LON SmartServer 2.0* Programming Tools is included on the *i.LON SmartServer 2.0* Programming Tools DVD (Echelon part number 72111-409), which you can order from your Echelon sales representative.

Installing the demo or full version of the *i.LON SmartServer 2.0* Programming Tools adds the following programs to your computer:

- *i.LON SmartServer 2.0* Programming Tool. A pre-configured Eclipse Development Kit that includes FPM template files, the FPM library, a tool for creating the C structures of user-defined UNVTs, a C++ compiler, and a CYGWIN environment. You must have the full version of the *i.LON SmartServer 2.0* Programming Tools to compile and upload FPMs to your SmartServer with the *i.LON SmartServer 2.0* Programming Tool.
- *i.LON SmartServer 2.0* LonWorks Interface Developer tool. A command line interface that converts a model file (.nc extension) to a device interface (XIF) file. You must create a XIF for your FPM in order to deploy it on your SmartServer. See Chapter 4 for more information on creating XIFs with this tool.
- *i.LON* License Generator. A tool for creating licenses that help protect your FPM application from piracy or unauthorized use. The *i.LON* License Generator includes the following three components:
  - The main executable (**iLONLicenseGen.exe**) that provides a user interface for entering the values used to generate an FPM license.
  - A sample license generator configuration file (an XML file named **iLONLicenseGenValuesSample.xml**) that demonstrates the structure of the *i.LON* License Generator user interface and provides sample pre-defined values.
  - A sample security DLL file (**LicenseSecurityHMACMD5.dll**) that takes the values entered in the *i.LON* License Generator user interface and creates an FPM license.

See Chapter 7 for more information on creating FPM application licenses.

You can install the full version of the *i.LON SmartServer 2.0* Programming Tools on a computer on which the demo version has not been installed, or you can upgrade a demo version of the *i.LON SmartServer 2.0* Programming Tools to the full version. The following section describes how to install the *i.LON SmartServer 2.0* Programming Tool for both scenarios. You can also upgrade your *i.LON SmartServer 2.0* Programming Tools as updates become available.

---

### *Installing i.LON SmartServer 2.0 Programming Tools*

To install the full version of the *i.LON SmartServer 2.0* Programming Tools, follow these steps:

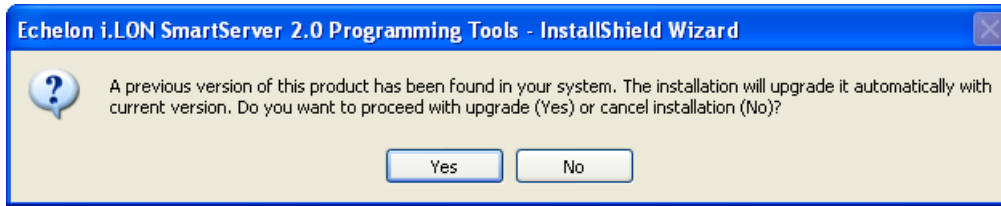
1. Insert the *i.LON SmartServer 2.0* Programming Tools DVD into your DVD-ROM drive. If your computer does not have a DVD-ROM, insert the *i.LON SmartServer 2.0* Programming Tools DVD on a network-accessible computer that has a DVD-ROM and copy the files on the DVD to a shared network drive. You can then copy the **LonWorks\iLON\Development** folder from the shared drive to your computer and install the *i.LON SmartServer 2.0* Programming Tools.
2. If the *i.LON SmartServer 2.0* setup application does not launch immediately, click **Start** on the taskbar and then click **Run**. Browse to the **setup.exe** file on the root directory of the *i.LON SmartServer 2.0* Programming Tools DVD and click **Open**. The **i.LON SmartServer 2.0 2.0** main menu opens.



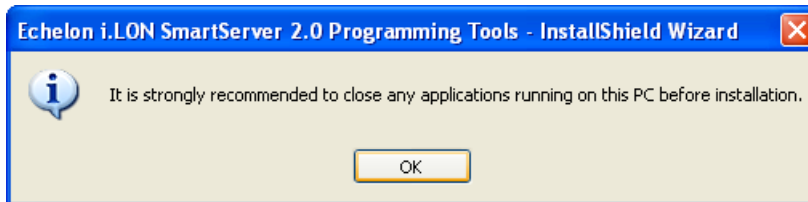
3. Click **Install Products**. The **Install Products** dialog opens.



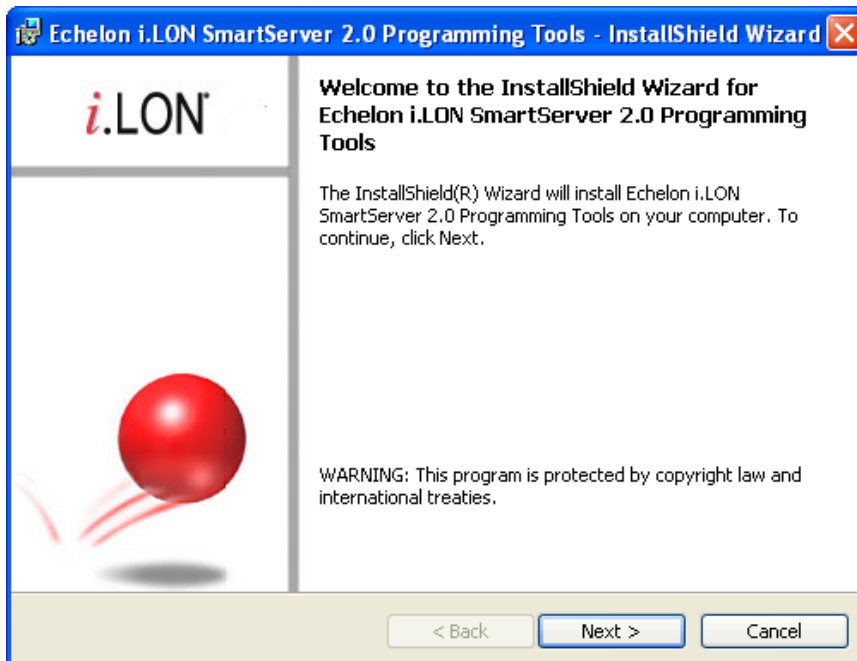
4. Click **Echelon i.LON SmartServer 2.0 Programming Tools**.
5. If a previous version of the *i.LON SmartServer 2.0 Programming Tools* (Release 4.0, 4.01, 4.02, or the demo version of Release 4.03) is installed on your computer, the following dialog opens prompting you to confirm that you want to upgrade to the *i.LON SmartServer 2.0 Programming Tools* software. Click **Yes** to upgrade.



6. A dialog opens prompting to close all applications currently running on your computer. Close any applications running on your computer, and then click **OK**.



7. The Echelon i.LON SmartServer 2.0 Programming Tools software installer opens.

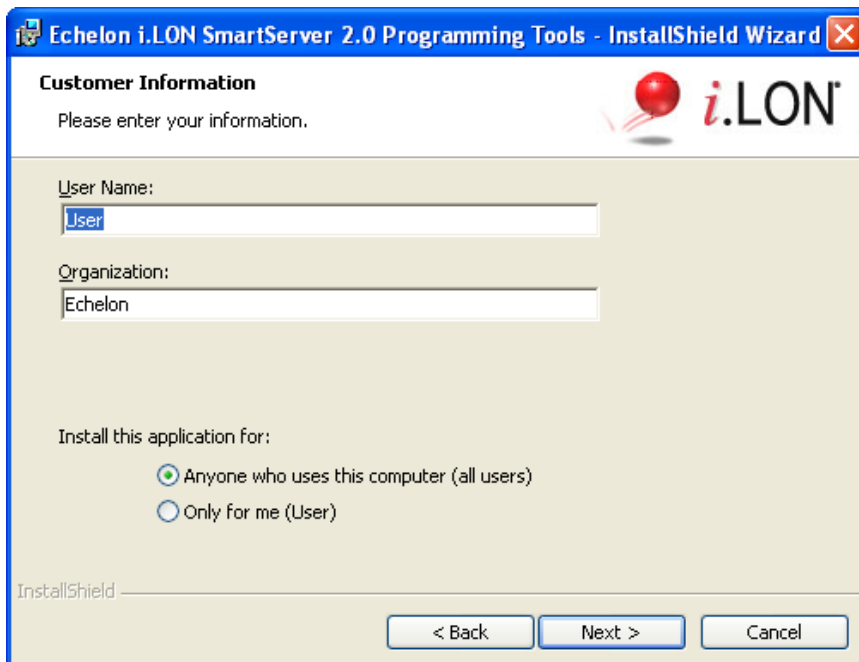


8. Read the information on the Welcome window and click **Next**. The License Agreement window appears.

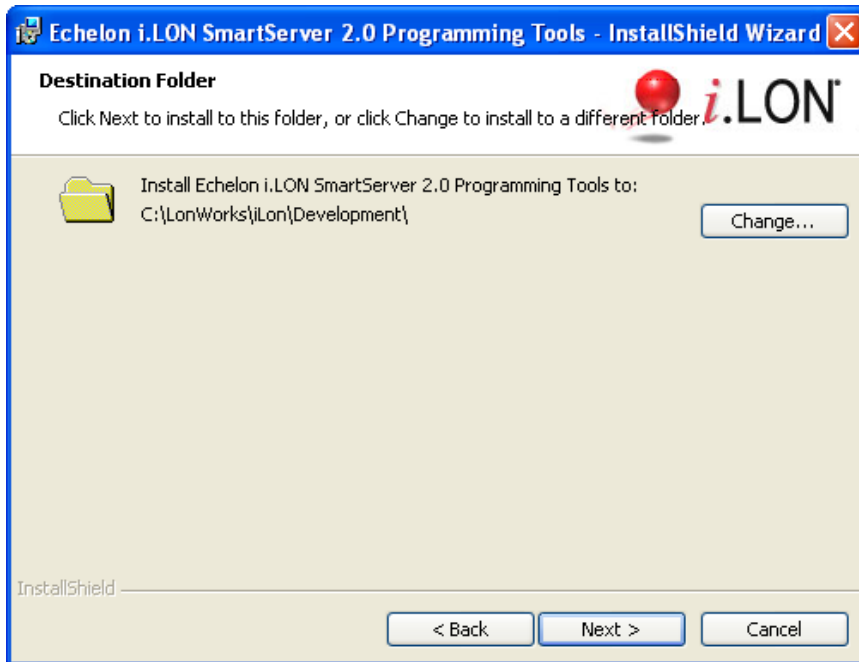




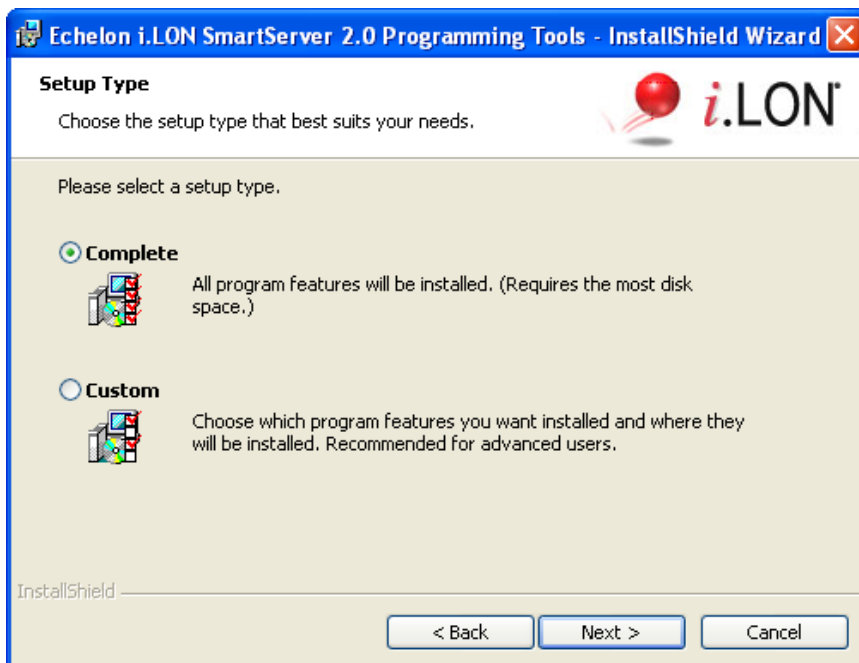
9. Read the license agreement (you can read a printed copy of this license agreement in Appendix E of the *i.LON SmartServer 2.0 User's Guide*). If you agree with the terms, click **Accept the Terms** and then click **Next**. The Customer Information window appears. .



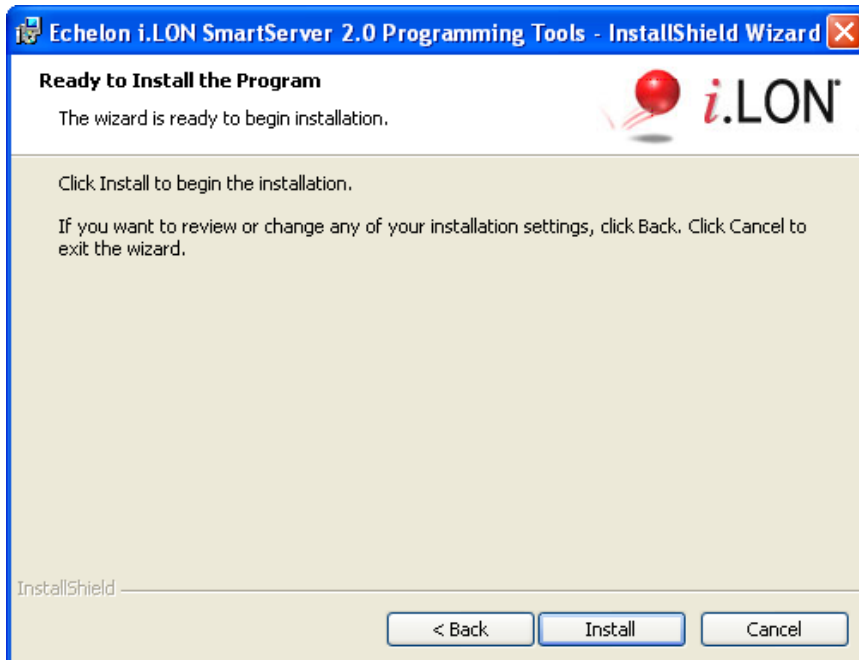
10. Enter your name and company name in the appropriate fields. The name and company may be entered automatically based on the user currently logged on and whether other Echelon products are installed on your computer. Click **Next**. The Destination Folder window opens.



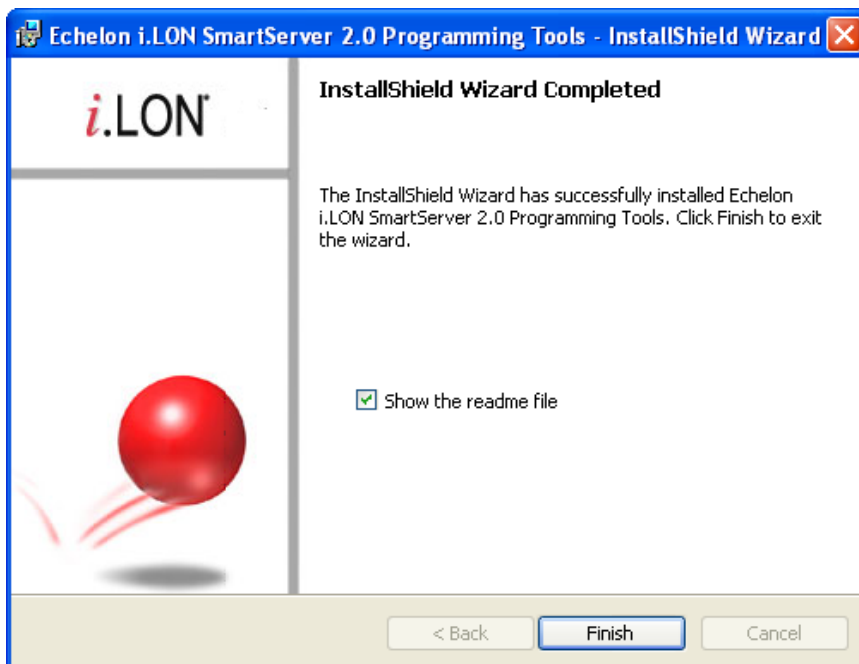
11. By default, the *i.LON* SmartServer 2.0 Programming Tools software will be installed in the **C:\LonWorks\iLON\Development** directory, or it will be installed in the **C:\Program Files\LonWorks\iLON\Development** directory if you have not previously installed any Echelon or LONMARK products. You can click **Change** to select a different destination folder. Click **Next**. The Setup Type window appears.



12. Select the type of installation to be performed. It is recommended that you select **Complete**. Click **Next**. The Ready to Install window appears.



13. Click **Install** to begin the installation. After the *i.LON SmartServer 2.0 Programming Tools* have been installed, a window appears stating that the installation has been completed successfully.



14. Click **Finish**. The *i.LON SmartServer 2.0 Programming Tools ReadMe* file appears. When you finish reading the ReadMe file, close the window.

---

## ***Upgrading the i.LON SmartServer 2.0 Programming Tool***

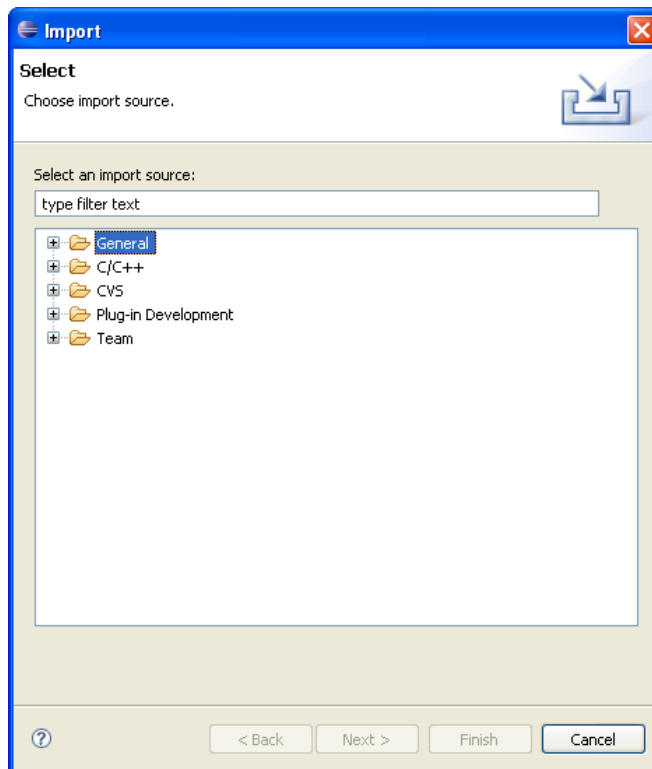
You can upgrade your *i.LON SmartServer 1.0 Programming Tools* to the *i.LON SmartServer 2.0 Programming Tools*, and you can upgrade your *i.LON SmartServer 2.0 Programming Tools* to newer versions as they become available. To upgrade your *i.LON SmartServer 2.0 Programming Tools*, you do the following:

1. Browse to the **LonWorks\iLON\Development\eclipse\workspace.fpm** folder on your computer. This folder contains your FPM projects and code. This folder is required if you later re-install the *i.LON SmartServer 2.0 Programming Tool* and need to import and modify your existing FPM projects or port code over to new FPM projects. See the next section, *Importing FPM Projects*, for more information on how to import existing FPM projects to an updated version of the *i.LON SmartServer 2.0 Programming Tool*.
2. Copy all your FPM projects and save them to the local drive of your computer, a USB drive, a floppy disk, another removable media, or a shared network drive with read/write permissions.
3. Uninstall the current version of the *i.LON SmartServer 2.0 Programming Tools* on your computer.
4. Install the newest version of the *i.LON SmartServer 2.0 Programming Tools* following the steps described in the previous section.
5. Import your existing FPM projects as described in the next section.
6. If you want to upgrade existing FPMs built with the *i.LON SmartServer 1.0 Programming Tool* (Release 4, Release 4.01, or Release 4.02 FPMs), convert them as described in *Converting FPM Projects to the Release 4.03 Configuration* later in this section.

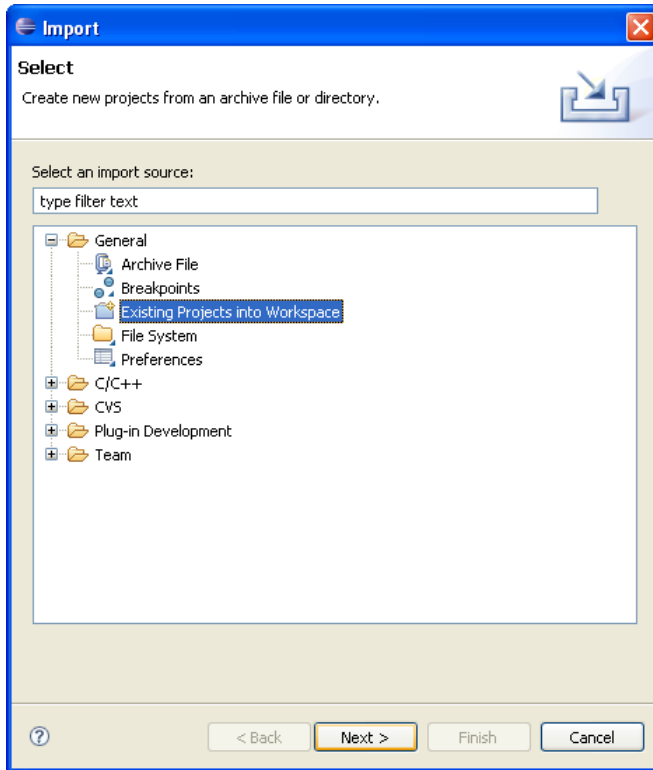
### *Importing FPM Projects*

After you upgrade the *i.LON SmartServer 2.0 Programming Tools* software, you can import your existing FPM projects into the upgraded *i.LON SmartServer 2.0 Programming Tool*. To do this, follow these steps:

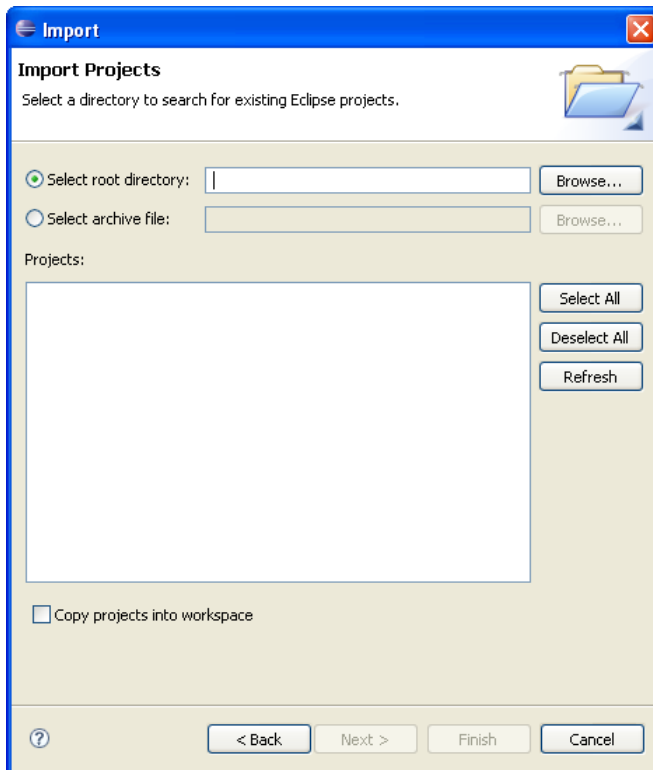
1. Start the *i.LON SmartServer 2.0 Programming Tool*. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0 Programming Tool* opens.
2. Click **File** and then click **Import**. The **Import** dialog opens with the **Select** window.



3. Expand the **General** folder, click **Existing Projects into Workspace**, and then click **Next**.



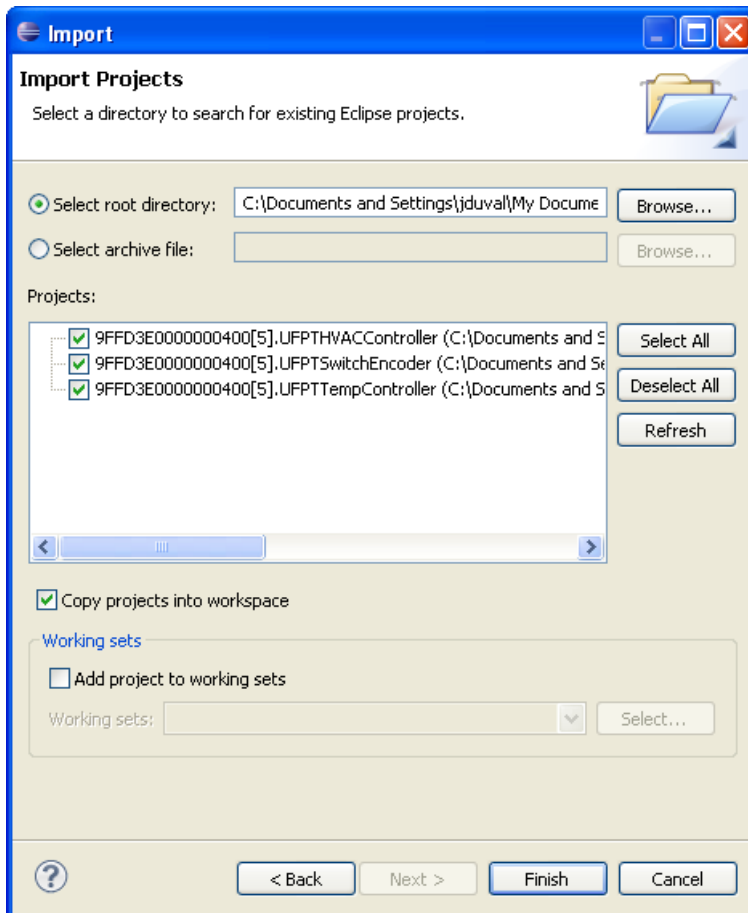
4. The Import Projects window opens.



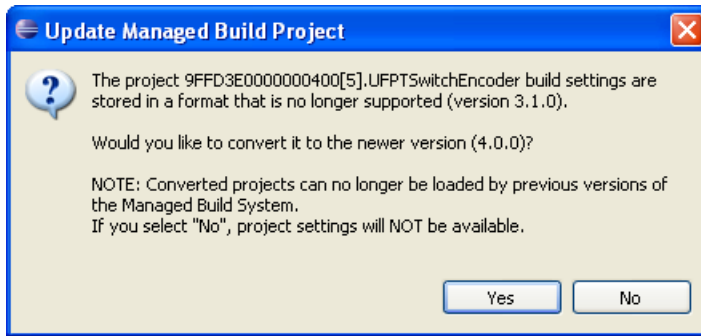
5. Click **Browse**. The **Browse for Folder** dialog opens. Browse to the location of the backup copy of the FPM project to be imported and then click **OK**.



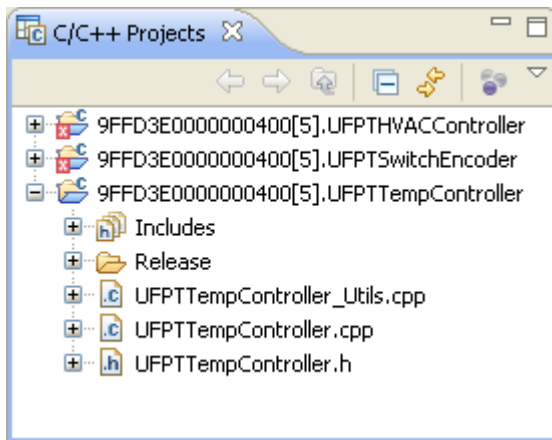
- The selected FPM projects are listed in the **Import Projects** dialog. Select the **Copy Projects into Workspace** check box.



- Click **Finish**.
- If the **Update Managed Build Project** dialog opens, click **Yes** to convert the build settings to the version 4.0 format



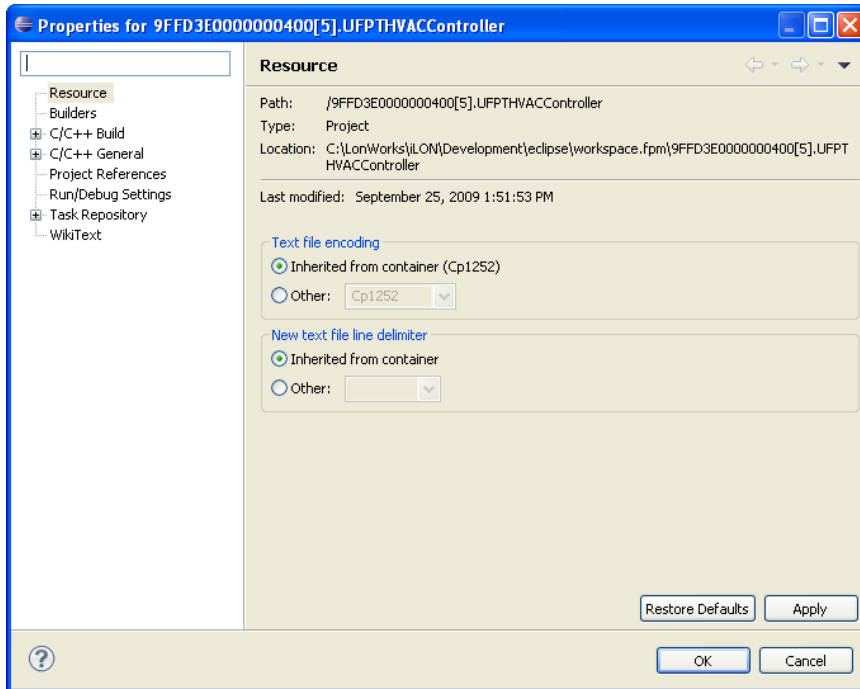
9. The selected FPM projects appear in the **C/C++ Projects View**.



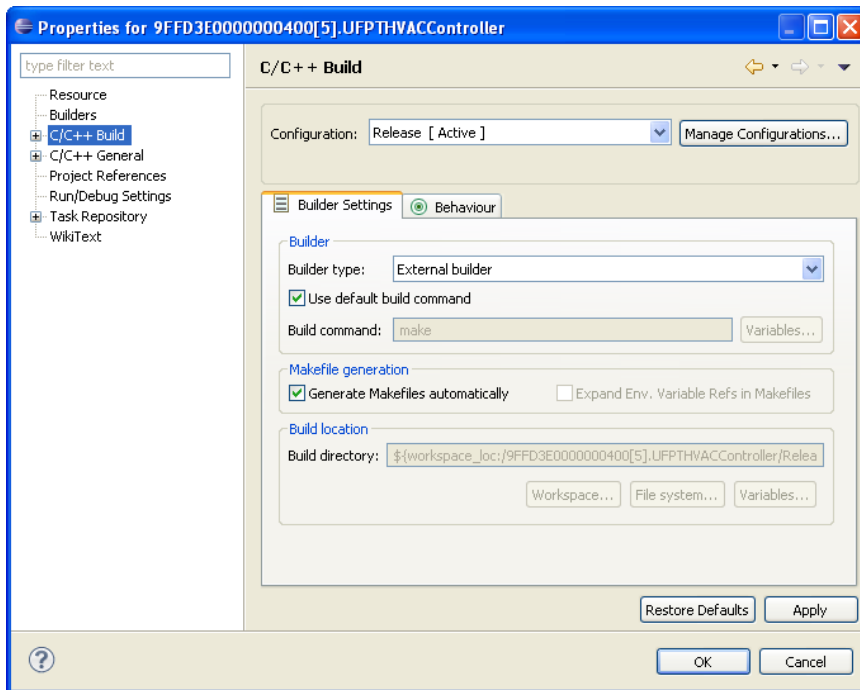
### *Converting FPM Projects to the Release 4.03 Configuration*

If you want to upgrade existing FPMs built with the *i.LON SmartServer 1.0 Programming Tool* (Release 4, Release 4.01, or Release 4.02 FPMs), you must first convert them to the Release 4.03 configuration. To do this, follow these steps:

1. Right-click the FPM project in the **C/C++ Projects** view, and click **Properties** on the shortcut menu, or click the FPM project, click **File**, and then click **Properties**. The **Properties** dialog for the FPM project opens.

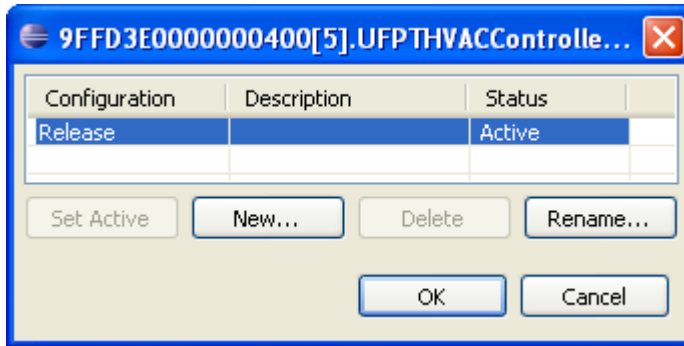


2. Click **C/C++ Build**.

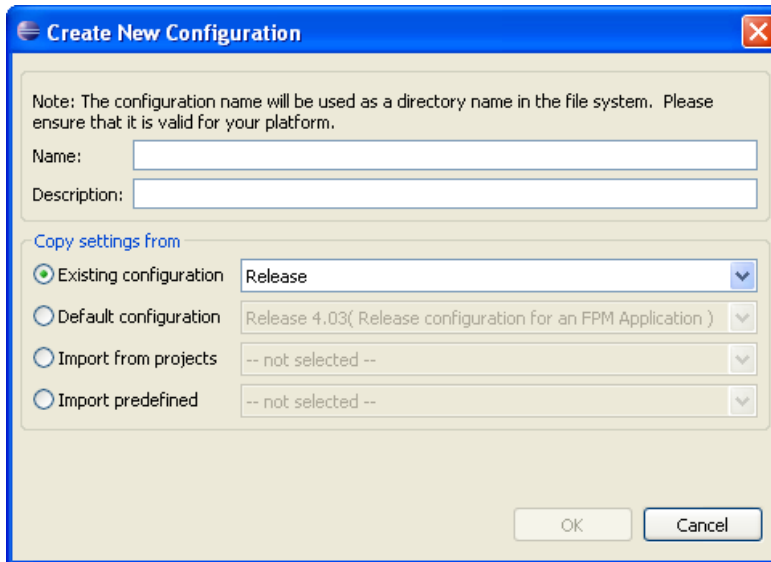


3. Click **Manage Configurations**. The **Manage Configurations** dialog opens.

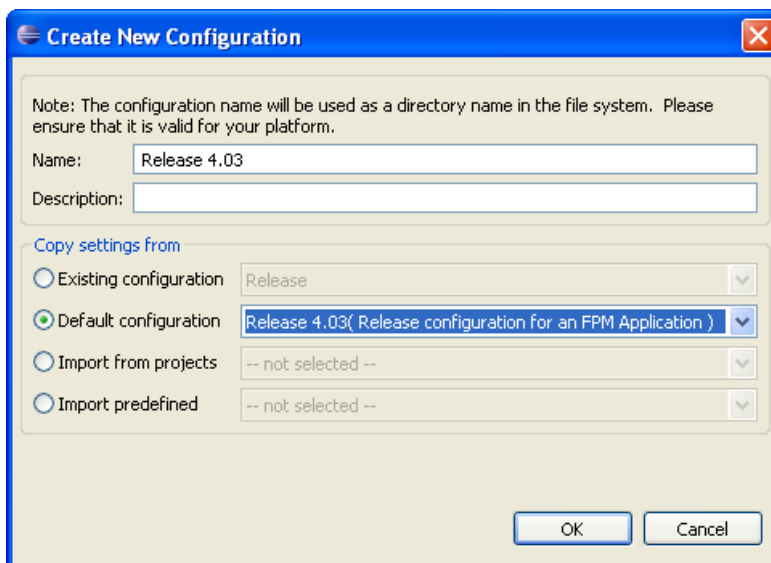




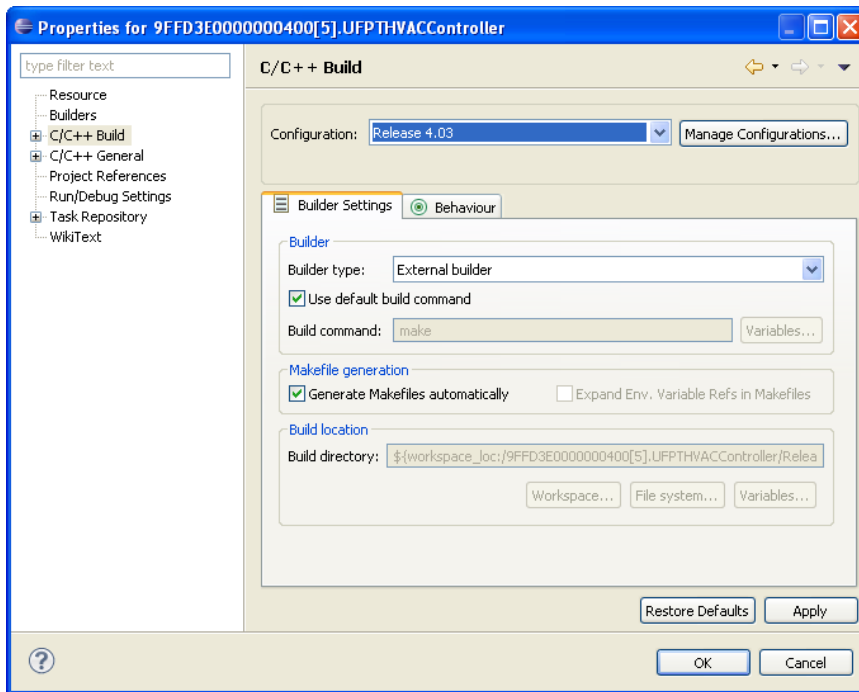
4. Click **New**. The **Create New Configurations** dialog opens.



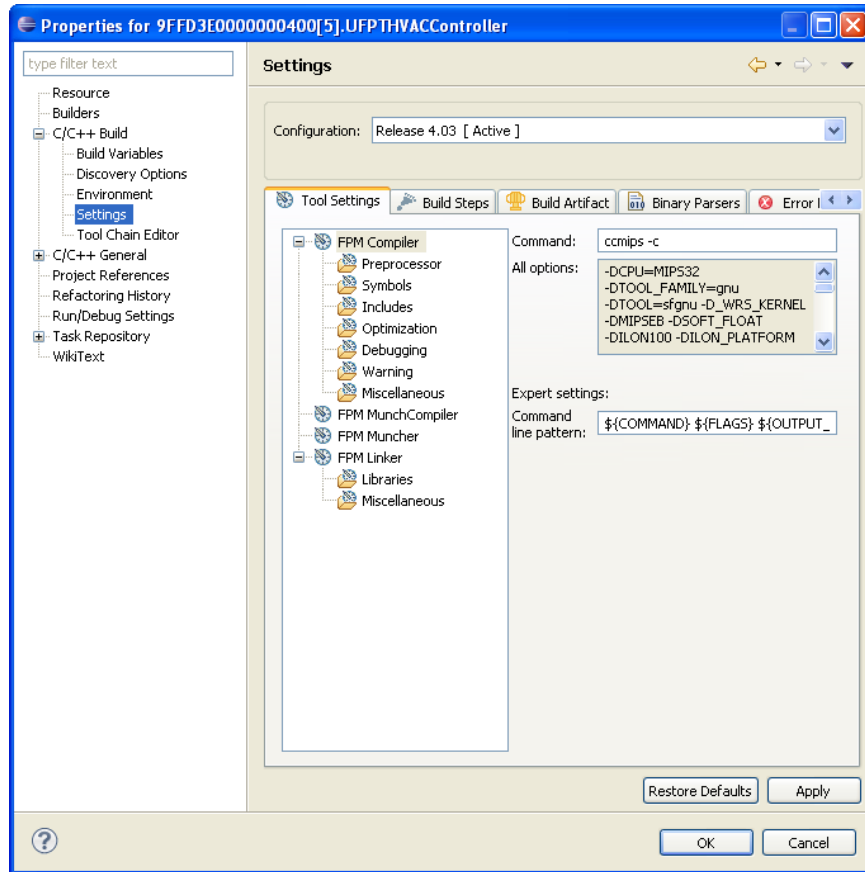
5. In the **Name** property, enter **Release 4.03**. If you are upgrading a debug configuration of an FPM, enter **Debug 4.03**.
6. In the **Default Configuration** property, select **Release 4.03**. If you are upgrading a debug configuration of an FPM, select **Debug 4.03**.



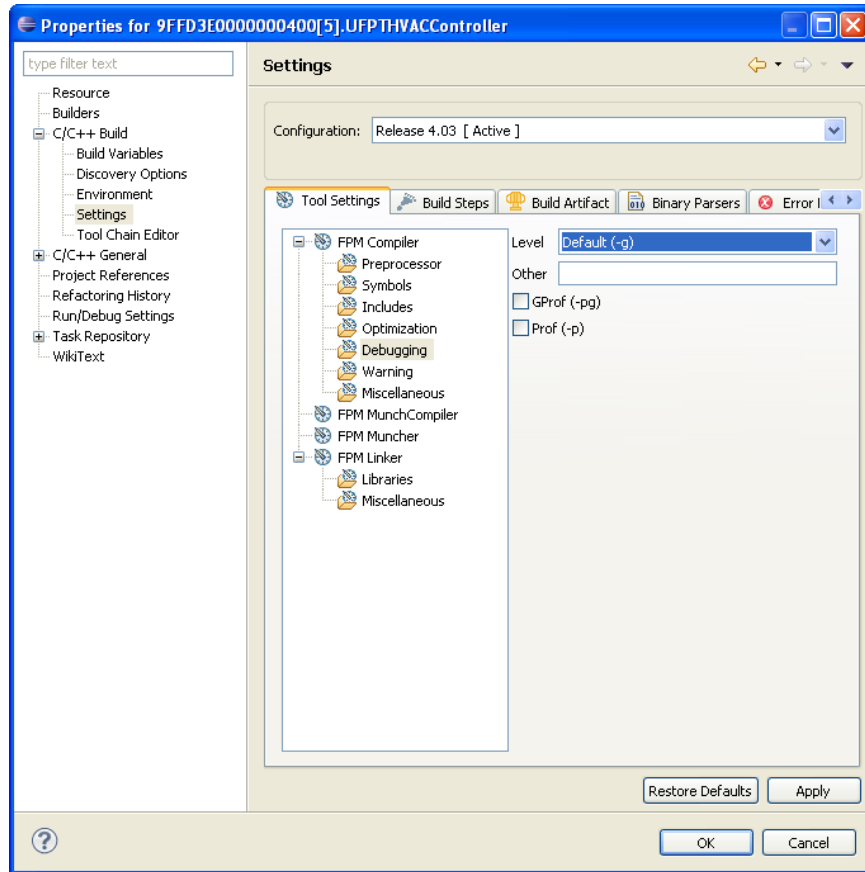
7. Click **OK** twice to return to the **C/C++ Build** window.
8. In the **Configuration** property, select **Release 4.03**. If you are upgrading a debug configuration of an FPM, select **Debug 4.03**.



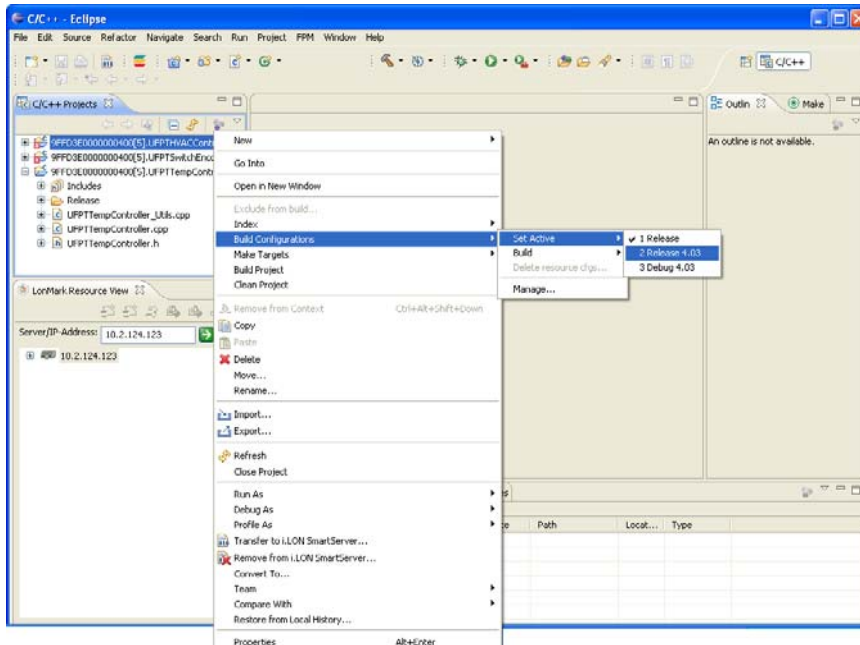
9. If you are upgrading a release configuration of an FPM, Click **OK** to return to the **C/C++ Projects** view, and then skip to step 10. If you are upgrading a debug configuration of an FPM, configure its settings following these steps:
  - a. Expand **C/C++ Build** and then click **Settings**.



- b. Click **Debugging** in the **Tool Settings** tab.
- c. In the **Level** box, select **Default (-g)**.



- d. Click **OK** to return to the **C/C++ Projects** view.
10. Set the build target for your FPM to the Release 4.03 or Debug 4.03 configuration. To do this, right-click the FPM project in the **C/C++ Projects** view, point to **Build Configurations**, point to **Set Active**, and then click **Release 4.03** or click **Debug 4.03** if you are upgrading a debug configuration of an FPM.



11. You can upgrade your FPM application or driver, rebuild it, and then upload it to your SmartServer 2.0.

**Note:** FPMs that have been upgraded to the Release 4.03 or Debug 4.03 configuration can only run on a SmartServer 2.0 (a SmartServer with the Release 4.03 image). You cannot run an upgraded FPM on a SmartServer 1.0 (a SmartServer with a Release 4.0, 4.01, or 4.02 image).

---

## *Uninstalling i.LON SmartServer 2.0 Programming Tools*

If you need to uninstall the *i.LON SmartServer 2.0 Programming Tools*, you should back up the LonWorks\iLON\Development\eclipse\workspace.fpm folder on your computer. This folder contains your FPM projects and code. This folder is required if you later re-install the *i.LON SmartServer 2.0 Programming Tools* and need to modify your existing FPM applications or port code over to new FPM projects.



## Creating FPM Templates

This chapter describes how to use the NodeBuilder Resource Editor to create user-defined functional profile templates (UFPTs). It explains how to upload your company's updated FPM resource file set to your SmartServer so that you can create an FPM project and begin writing your FPMs.

---

## Creating FPM Templates Overview

Before you can begin writing an FPM, you need to create its user-defined functional profile template (UFPT) using NodeBuilder Resource Editor 3.13. Functional profile templates are LONMARK specifications that enable you to specify the functionality required for a device. A functional profile template defines the set of network variables and configuration properties within a functional block that collectively perform a single device function. A functional profile template is defined in a resource file with an **.fpt** extension.

For an FPM specifically, a functional profile template defines the data point types to be declared in an FPM application or FPM driver.

To create a functional profile template for an FPM, you do the following:

1. Create a UFPT. You can create a new UFPT, or you can create one by copying an existing SFPT to your resource file set.
2. Add the standard and user-defined network variable types (SNVTs and UNVTs) and configuration property types (SCPTs and UCPTs) to which the FPM will read and write.
3. Generate the updated FPM resource file set in which the template was created and copy it to the flash disk of your SmartServer.

For more detailed information on using the NodeBuilder Resource Editor to edit resource files and create functional profile templates, network variables, and configuration properties, see the *NodeBuilder Resource Editor User's Guide*.

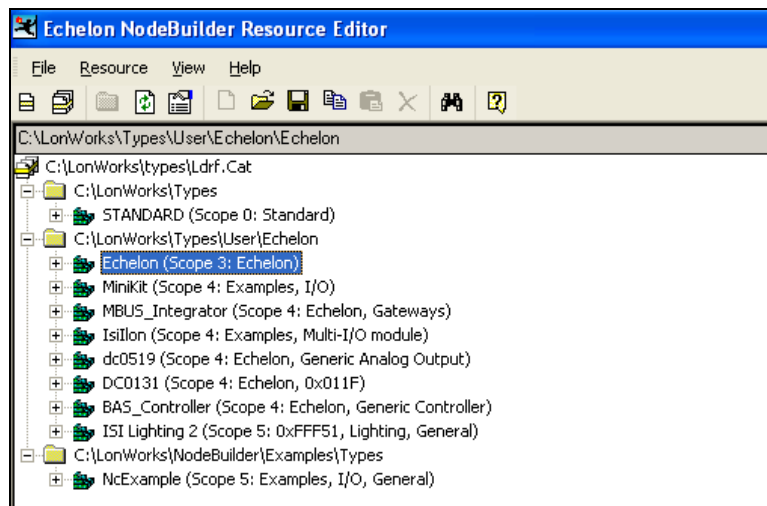
**Note:** If NodeBuilder Resource Editor is not installed on your computer, you can install NodeBuilder Resource Editor 3.14.02 from the *i.LON SmartServer 2.0 DVD* or the *i.LON SmartServer 2.0 Programming Tools DVD*. See the *i.LON SmartServer 2.0 User's Guide* for more information on installing the NodeBuilder Resource Editor.

---

## Creating User-Defined Functional Profile Templates

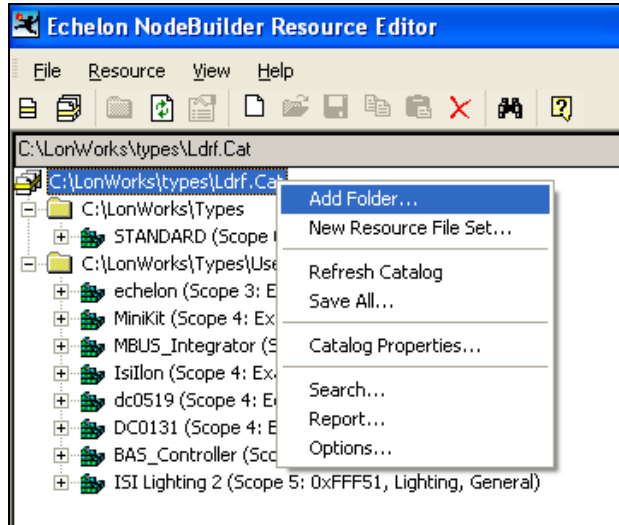
You can create a new UFPT in any scope 3, 4, 5, or 6 resource file set. Do not create or modify a functional profile in a resource file set that does not have your manufacturer ID or is one that you do manage. To create a new UFPT, follow these steps:

1. Start the NodeBuilder Resource Editor. To do this, click **Start**, point to **Programs**, point to **Echelon NodeBuilder Resources**, and then click **NodeBuilder Resource Editor**. The **Echelon NodeBuilder Resource Editor** opens.

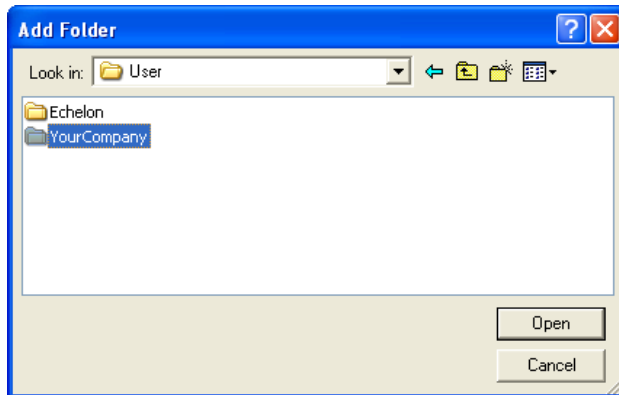




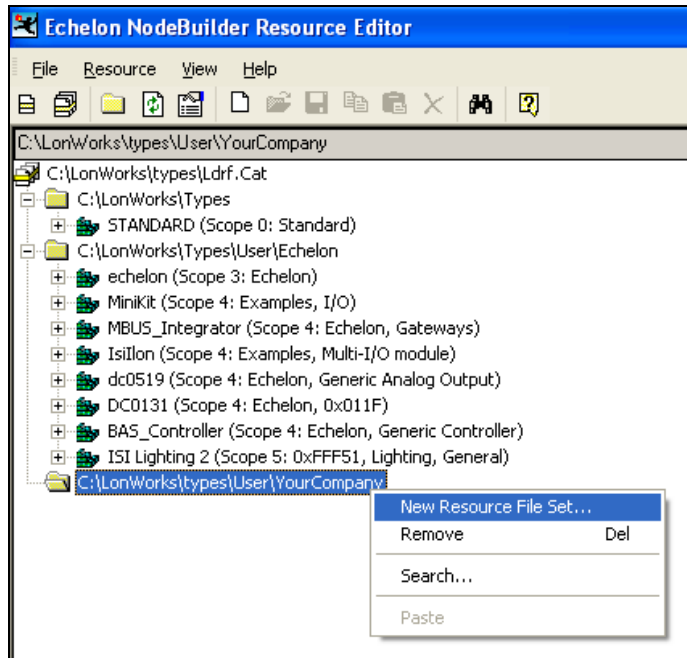
2. Create a <YourCompany> folder under the LonWorks\types\user directory on your computer if one doesn't already exist. To do this, follow these steps:
  - a. Right-click the **LonWorks\types\Ldrf.Cat** file and then click **Add Folder** on the shortcut menu.



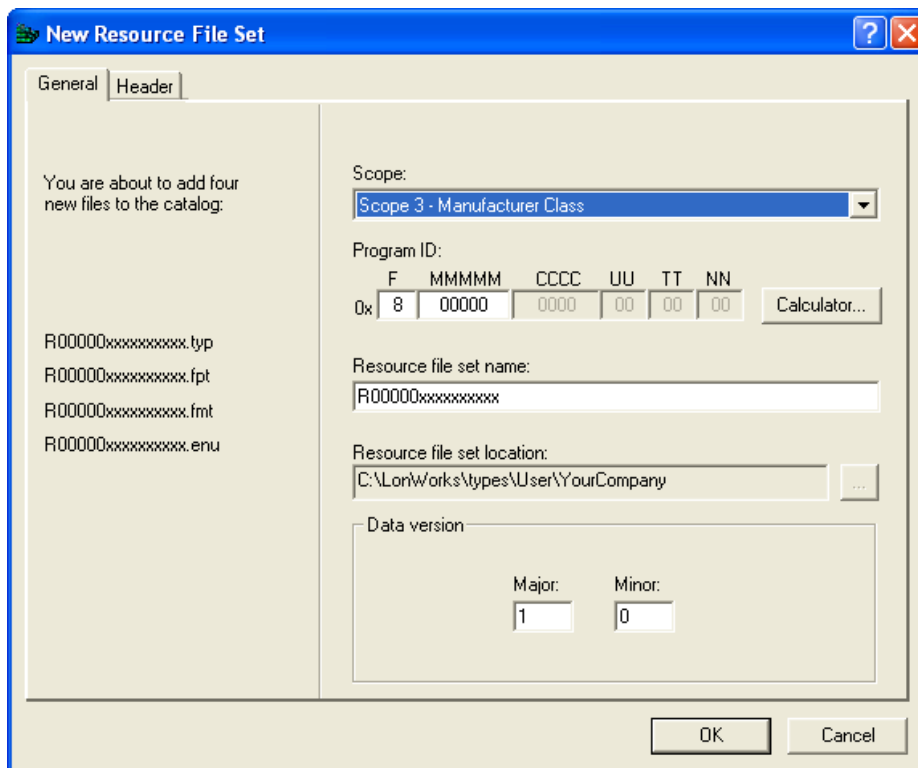
- b. Open the User folder, create a new folder named *YourCompany*, and then click **Open** twice.



- c. The *YourCompany* folder appears at the bottom of the resource catalog view.
3. Create a new resource file set for your company. If you plan on integrating your FPM applications with an LNS application such as the LonMaker tool, you need to create a scope 5 resource file set. To create a new resource file set, follow these steps:
  - a. Right-click your company's resource file set and click **New Resource File Set** on the shortcut menu.



- b. The **New Resource File Set** dialog opens.



- c. If you plan on integrating your FPM applications with an LNS application such as the LonMaker tool, you should select **5** in the **Scope** box (this sets the scope to device class, manufacturer, usage, and channel type).
- d. In the manufacturer (**MMMMM**) field of the **Program ID** box, enter your 5-digit manufacturer ID in hexadecimal format.

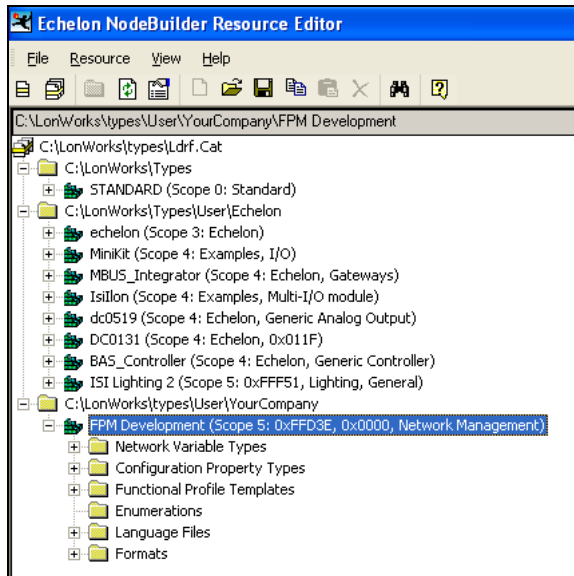
**Note:** If your company does not have a manufacturer ID, you can get a temporary manufacturer ID from LonMark at [www.lonmark.org/mid](http://www.lonmark.org/mid). In addition, if your company has many FPM developers, it is recommended that you request temporary manufacturer IDs for them. After you obtain your temporary manufacturer ID, you can enter it in the **MMMMM** field of the **Program ID** box.

- e. In the format (**F**) field of the **Program ID** box, enter 9 (this sets the Standard Development Program ID flag).
- f. In the channel (**TT**) field of the **Program ID** box, enter **04** if you have an FT-10 model of the SmartServer or enter **10** if you have a PL-20 model of the SmartServer.
- g. In the **Resource File Set Name** box, enter “FPM Development”, “FPM Examples”, or something comparable.

The screenshot shows the 'New Resource File Set' dialog box with the following fields and values:

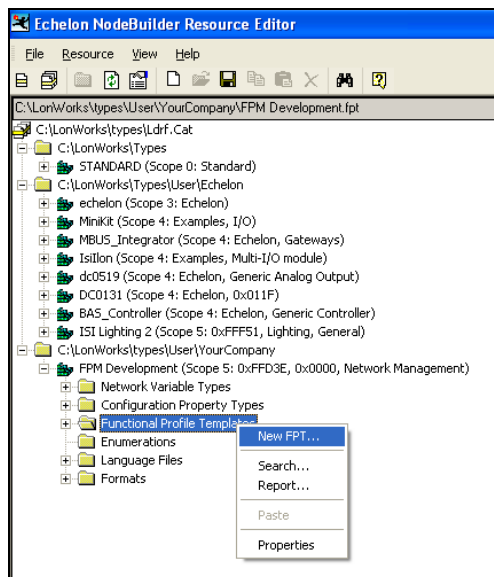
- Scope:** Scope 5 - Manufacturer, Device Class, Usage and Channel Type
- Program ID:**
  - F: 9
  - MMMMM: FFD3E
  - CCCC: 0000
  - UU: 00
  - TT: 04
  - NN: 00
- Resource file set name:** FPM Development
- Resource file set location:** C:\LonWorks\types\User\YourCompany
- Data version:**
  - Major: 1
  - Minor: 0

- h. Click **OK**.
4. Your company’s FPM resource file set is created and added to the resource catalog under the <Your Company> folder in the LonWorks\types\user directory.




5. Create a UFPT. You can create a new UFPT, or you can create one by copying an existing SFPT to your resource file set. To create a UFPT from an SFPT, skip to step 7. To create a new UFPT, follow these steps:

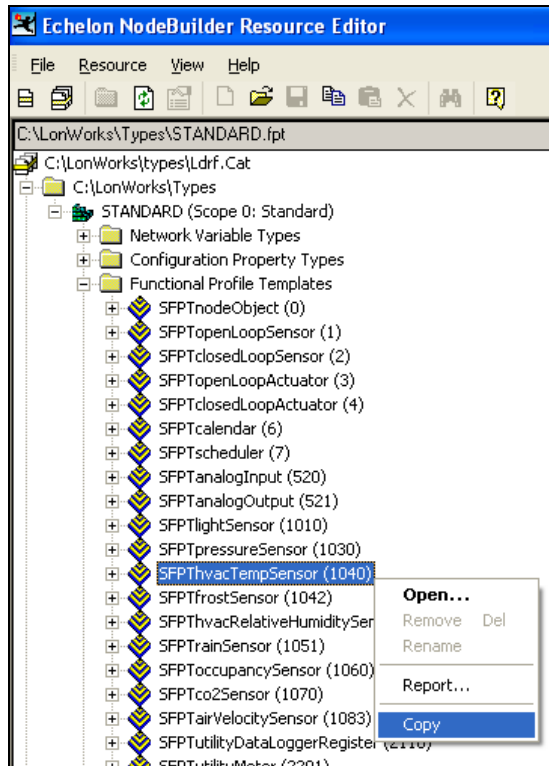
- a. Expand the folder containing your company's FPM resource file set, right-click the **Functional Profile Template** folder, and then click **New FPT** on the shortcut menu.



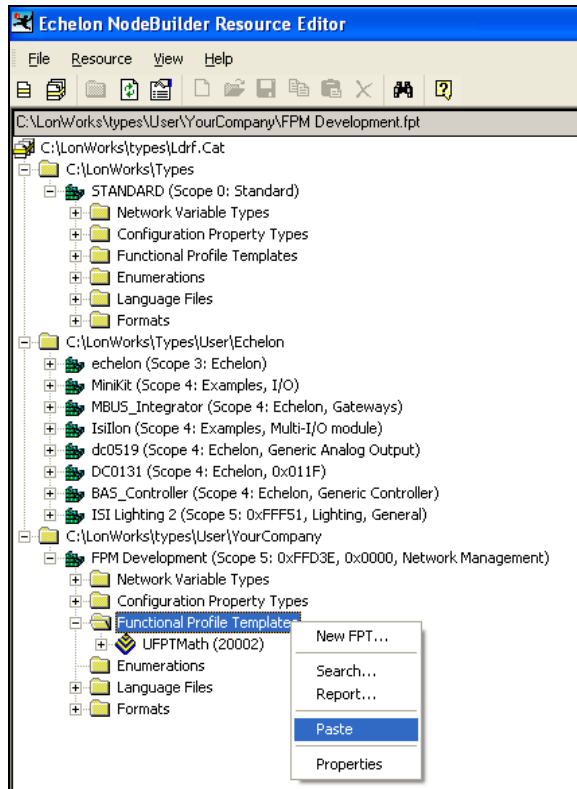
**Note:** Do not expand a resource file set that does not have your unique program ID or is one that you do not manage.

- b. A functional profile template icon (  ) with a default name of **UFPT<index>** (<key>) is added to the **Functional Profile Templates** folder.
- c. Enter a meaningful name for the new UFPT. By convention, the functional profile name should indicate the application set of the profile (for example, "UFPTHVACController"). The name must start with "UFPT", and by convention, there is no underscore following UFPT; the first letter after UFPT is upper case (for SmartServer embedded applications only); and the name uses mixed case. Functional profile names are limited to 64 characters, including the "UFPT" prefix, and cannot contain spaces or dollar characters.

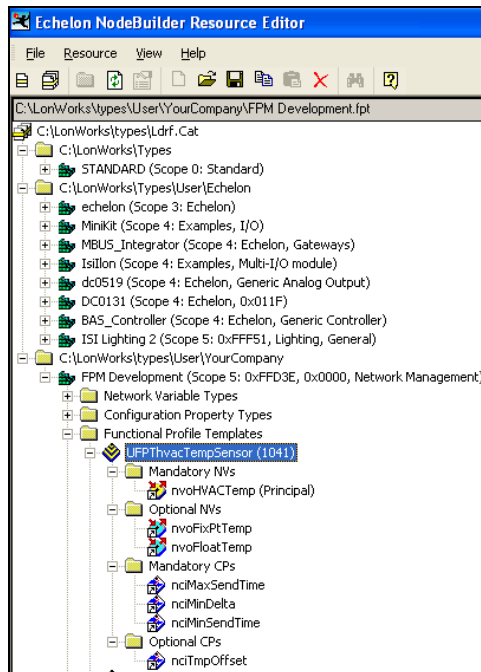
6. To create a UFPT from an SFPT, follow these steps:
  - a. Expand the **STANDARD** resource file set under the LonWorks\Types folder, expand the **Functional Profile Templates** folder to show all the SFPTs in the folder, right-click a SFPT, and then click **Copy** on the shortcut menu.



- b. Right-click the **Functional Profile Templates** folder in your company's FPM resource file set and click **Paste** on the shortcut menu.



- c. You can then add, delete, and edit the network variable and configuration property members in the SFPT to fit your FPM application or driver.

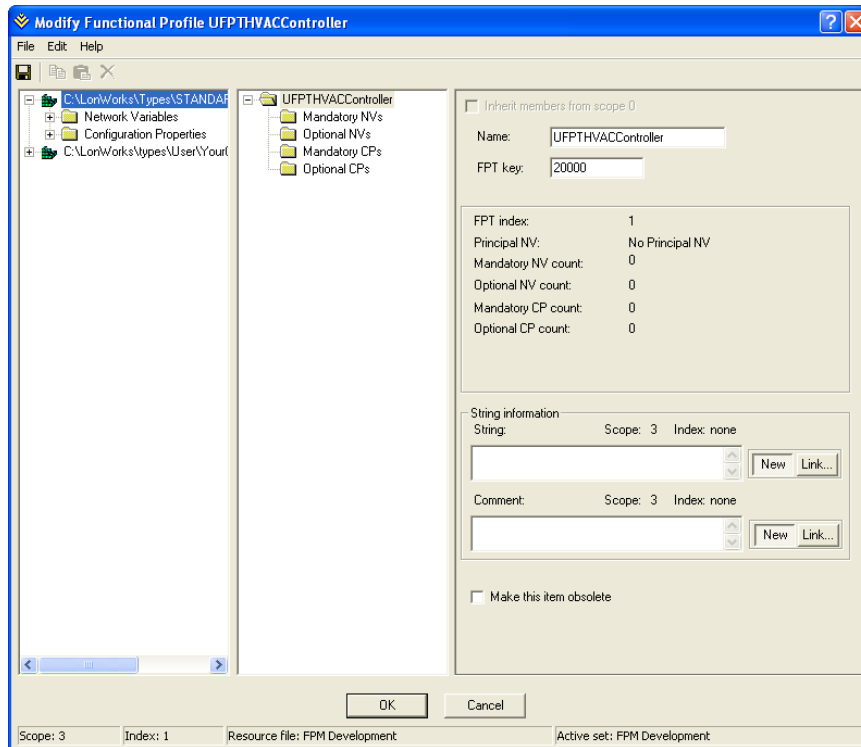


## *Adding Network Variable and Configuration Property Types*

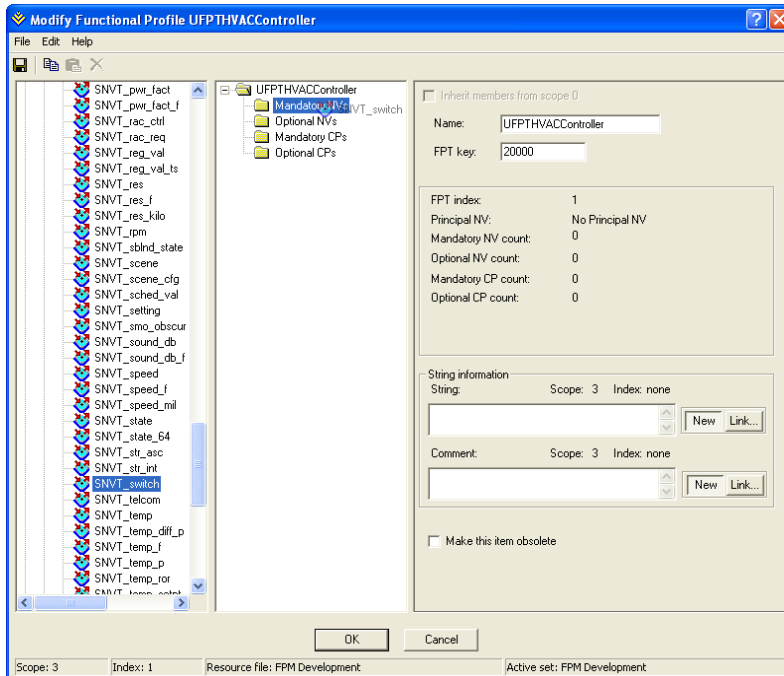
You can add network variable and configuration property types to your UFPT. You can add types that are defined in the standard scope 0 resource file set and types defined in your company's FPM

resource file set (the resource file set that has your manufacturer ID). To add network variable and configuration property to your UFPT, follow these steps:

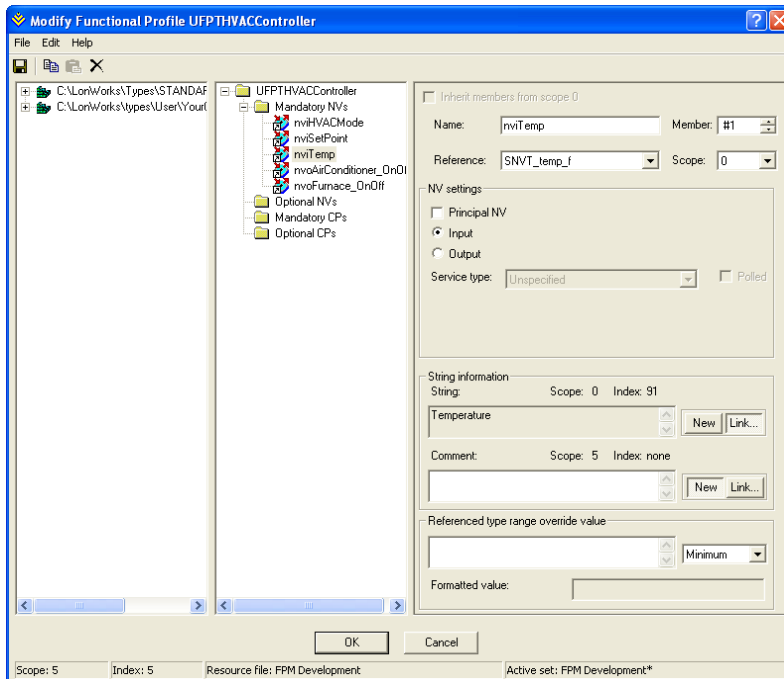
1. Double-click the new UFPT, or right-click it and click **Open** on the shortcut menu. The **Modify Functional Profile** dialog opens.



2. Expand the **LonWorks\types\STANDARD** resource file set (or expand your company's FPM resource file set to add user-defined data types), expand the **Network Variables** or **Configuration Properties** folder in the Resource (left) pane.
3. Click and drag the target data type to the **Mandatory NVs** or **Mandatory CPs** folder in the Profile (center) pane if the device interface used by the FPM must implement the target data type, or drag the target data type to the **Optional NVs** or **Optional CPs** folder if the FPM device interface has the option of implementing or not implementing the target data type.



- The selected data type appears below the folder to which it was added in the Profile (center) pane, and the selected data type can be edited in the Properties (right) pane.



- If you added a network variable to the functional profile template, enter the following information in the Properties (right) pane and then skip to step 7:



*Name* Enter the name of the network variable within the functional profile template. The name of the network variable may contain letters, digits, and underscore characters, but it cannot start with a digit.

Optionally, you can insert “nvi” and “nvo” in front of input and output network variable names, respectively, to simplify the identification of the input and output network variables in the functional profile template. For example, you can name a **SNVT\_temp** input data point “nviSetPoint”, or you can name a **SNVT\_switch** output data point “nvoLamp\_OnOff”.

If you do not insert an “nvi” or “nvo” prefix, the name of the data point should start with a capital letter and use mixed case. For example, you can name a **SNVT\_temp** input data point “SetPoint”, or you can name a **SNVT\_switch** output data point “Lamp\_OnOff”.

### NV Settings

*Principal NV* Designates this network variable as the principal network variable of the functional profile template. Each functional profile template may have one principal network variable. The principal network variable is used to determine the type of configuration properties with inherited types that apply to the functional profile template.

*Input/Output* Specify whether the network variable is an **Input** or **Output** network variable

6. If you added a configuration property to the functional profile template, enter the following information in the Properties (right) pane:

*Name* Enter the name of the configuration property within the functional profile template. The name of the configuration property may contain letters, digits, and underscore characters, but it cannot start with a digit. Optionally, you can insert “nci” in front of configuration property names to simplify the identification of the configuration properties within the functional profile template.

### CP Settings

*Array Implementation* Specify whether the configuration property within the functional profile template can be implemented as an array. You have the following three choices:

- **Prevent.** Functional blocks created using this functional profile template cannot implement this configuration property as an array. If you select this option, the **Min Array Size** and **Max Array Size** properties are unavailable. This is the default, and it applies to all functional profiles created prior to NodeBuilder 3.1.
- **Permit.** Functional blocks created using this functional profile template can implement this configuration property as an array at the discretion of the implementer. If you select this option, set the **Max Array Size** to limit the maximum size of the array. The **Min Array Size** property is unavailable.
- **Require.** Functional blocks created using this functional profile template must implement this configuration property as an array. If you select this option, set the **Max Array Size** and **Min Array Size** properties to limit the maximum and minimum size of the array.

*CP Settings* Select the following configuration property flags for the scenarios in which the configuration property can be changed. See the LONMARK

Application Layer Interoperability Guidelines for more information about configuration property restriction flags.

- **const\_flag**. The value of the configuration property cannot be changed.  
**Note:** When you deploy the FPM on the SmartServer, the configuration property is automatically set to be persistent. This means that the configuration property's default value is updated to match the current value stored in the configuration property, and that configuration property's current value persists through reboots.
- **device\_spec\_flag**. The value of the configuration property is always read from the device and can be modified independent of the LNS database.
- **mfg\_flag**. The configuration property value can only be changed when the device is being licensed.
- **obj\_disabl\_flag**. The device must be disabled for the configuration property value to be changed.
- **offline\_flag**. The device must be offline for the configuration property value to be changed.
- **reset\_flag**. The device is reset after the configuration property value is changed.

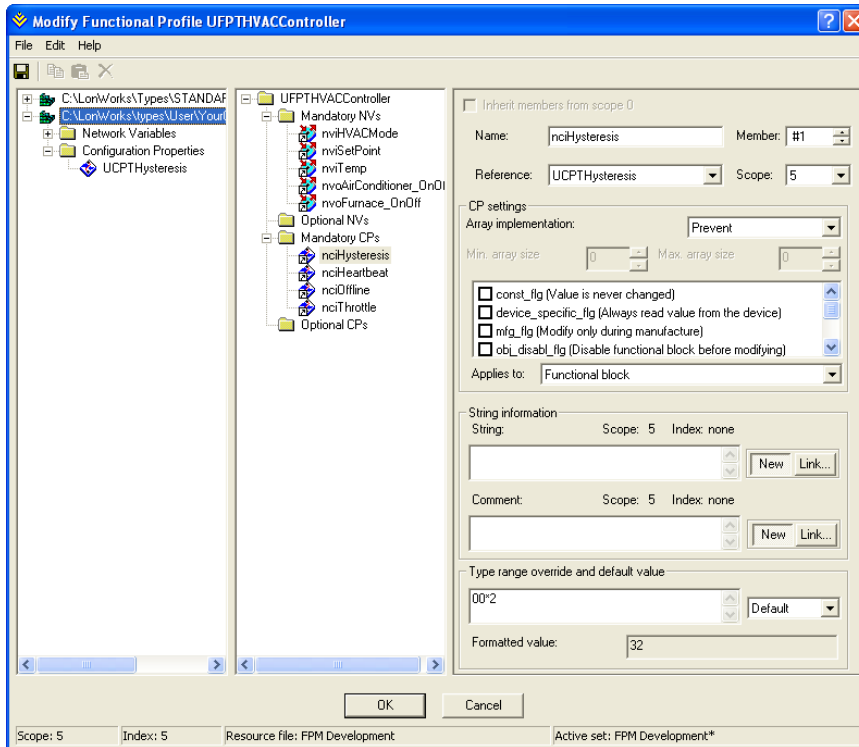
When a functional block implements a profile, each of the implemented member configuration properties must specify at least those restriction flags that are set in the profile. Restriction flags that are not set in the profile may be set by the implementing property, unless this would cause an ambiguous restriction flag set.

*Applies To*

Specify the scope of the configuration property. The configuration property can apply to the entire **Functional Block** or a network variable within the functional profile template.

If the configuration property applies to the functional block, and the functional block implements an inheriting type, the property will derive its type from the principal network variable. A principal network variable must be defined in this case.

If a configuration property is to apply to a specific network variable, select the network variable from the **Applies To** list.



7. Repeat steps 3–6 for each data type to be added to the functional profile template.
8. Click **OK**.

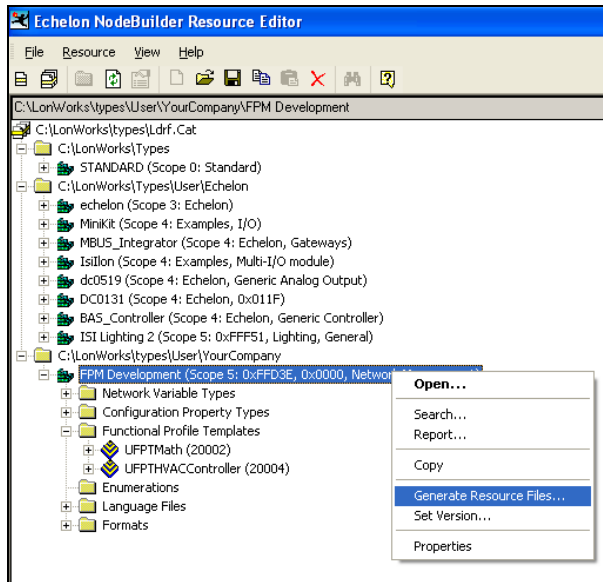
---

## *Generating and Copying the Updated FPM Resource File Set*

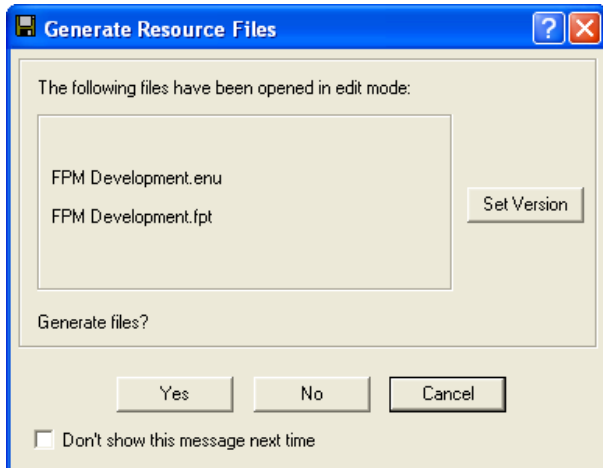
Once you create a UFPT and add all the required network variable and configuration property members to it, you can generate the updated FPM resource file set in which the UFPT was created. After you generate the updated FPM resource file set, you can copy it to the flash disk of your SmartServer.

To generate the updated FPM resource file set and copy it to your SmartServer, follow these steps:

1. Right-click your company's FPM resource file set, and then click **Generate Resources Set** on the shortcut menu.



2. The **Generate Resources Set** dialog opens.



3. Click **Yes** to generate the updated FPM resource file set.
4. Use FTP to access the root/lonworks/types/user directory on the flash disk of your SmartServer. Create a User/<YourCompany> folder if it does not already exist in the root/lonworks/types directory.
5. Browse to the LonWorks\Types\User\<YourCompany> folder on your computer, and then copy your company's **.ENU**, **.fmt**, **.fpt**, **.ls**, and **.typ** files to the root/lonworks/types/user/<YourCompany> directory on the SmartServer.
6. Repeat step 5 for each SmartServer on which the FPM is to be used. Your company's resource files must be installed on a SmartServer in order to create a functional block representing the FPM application on the SmartServer.
7. If you are using a static device interface (XIF) for your FPM, proceed to Chapter 4 to create a model file for your FPM, convert the model file to a XIF file, and upload the XIF file to your SmartServer. If you are using a dynamic XIF for your FPM, proceed to Chapter 5 to create a new FPM project from the UFPT and begin writing the FPM.

# Creating FPM Device Interface (XIF) Files

This chapter describes how to write a model file that declares the network variables and configuration properties in your FPM and a functional block implementing an instance of the UFPT used by your FPM. It explains how to use the *i.LON SmartServer 2.0 LONWORKS Interface Developer* tool to convert your model file to a device interface (XIF) file and how to copy the XIF to your SmartServer.

---

## Creating FPM Device Interface (XIF) Files Overview

In order to deploy an FPM application on the SmartServer, you need to create a device interface (XIF file extension). The XIF exposes the logical interface of your FPM application so that network tools such as the SmartServer and the LonMaker tool can manage it. The XIF specifies the number and types of functional blocks, and the number, types, and directions of the network variables and configuration properties in your FPM application. Note that the SmartServer only recognizes the text file version of the XIF (.xif extension).

To create a device interface (XIF) file for your FPM application, you do the following:

1. Create a model file (.nc extension) with a text or programming editor such as Notepad. In the model file, you declare all the network variables and configuration properties in the UFPT used by your FPM application, and you declare a functional block that implements an instance of that UFPT.
2. Generate a XIF from your model file using the *i.LON SmartServer 2.0 LonWorks Interface Developer* tool. Copy the XIF (.xif extension) to the root/lonWorks/Import/<YourCompany> folder on the SmartServer flash disk.

---

### Creating a Model File

The model file uses the Neuron C programming language to describe the functional blocks, network variables, and configuration properties in an FPM application. You do not need to be proficient in Neuron C to create a model file for an FPM because the model file does not include executable code. The *i.LON SmartServer 2.0 Programming Tools* includes a command line interface called the *i.LON SmartServer 2.0 LonWorks Interface Developer* tool that converts model files to XIFs. Note that the model file uses Neuron C Version 2.1 declaration syntax.

You can use any of the following methods to create a model file:

- Manually create a model file. A model file is a text file that you can create with any text or programming editor such as Notepad. This section describes the basic Neuron C statements required to declare network variables, configuration properties, and functional blocks in your model file.
- Reuse existing Neuron C code. You can reuse an existing Neuron C application that was originally written for a Neuron Chip or a Smart Transceiver as a model file. The *i.LON SmartServer 2.0 LonWorks Interface Developer* tool uses only the device interface declarations from a Neuron C application program, and ignores all other code. You might have to delete some code from an existing Neuron C application program, or exclude this code using conditional compilation.
- Automatically generate a model file. You can use the NodeBuilder Code Wizard, which is included with Release 3 or later of the NodeBuilder Development Tool, to automatically generate a model file. Using the NodeBuilder Code Wizard, you can define your device interface by dragging functional profiles and type definitions from a graphical view of your resource catalog to a graphical view of your device interface, and refine them using a convenient graphical user interface. When you complete the device interface definition, click the **Generate Code and Exit** button to automatically generate your model file. You can then use the main file produced by the NodeBuilder Code Wizard as your model file. Note that the NodeBuilder Code Wizard is not included with the *i.LON SmartServer 2.0 Programming Tools*, and it must be licensed separately. See the *NodeBuilder User's Guide* for details about using the NodeBuilder Code Wizard.

### Declaring Network Variables

A *network variable* is a data item that a device application expects to get from other devices on a network (an *input network variable*) or expects to make available to other devices on a network (an *output network variable*).

You must declare all the mandatory network variables in the UFPT you created for your FPM. You may declare none to all of the optional network variables in the UFPT.

### SYNTAX

You can declare a network variable in your model file using the following syntax:

```
network input || output type identifier;
```

The **network** keyword declares a network variable of a specific *type* with a specific *identifier*.

The **input** and **output** keywords define the direction of the network variable. The specified direction must match the one defined for the referenced network variable in the UFPT.

The **type** property corresponds to the standard or user-defined network variable type (SNVT or UNVT) used by the network variable. The specified data type must match the one defined for the referenced network variable in the UFPT.

The **identifier** property is a reference to the network variable in the UFPT. The name specified in this property is the one that will be used by network tools such as the SmartServer and the LonMaker tool for the referenced network variable. The maximum length of the identifier is 16 characters.

### EXAMPLES

The follow example demonstrates how to create input and output network variables in your model file:

```
network input SNVT_temp_f nviTemp;  
network input SNVT_temp_f nviSetPoint;  
network input SNVT_hvac_mode nviHVACMode;
```

```
network output SNVT_switch nvoAC_OnOff;  
network output SNVT_switch nvoFurnace_OnOff;  
network output SNVT_str_asc nvoStatus;
```

By convention, input network variable names have an *nvi* prefix, and output network variables have an *nvo* prefix.

For more information on declaring network variables, see Chapter 3 of the *Neuron C Programmer's Guide*.

### Declaring Configuration Properties

A *configuration property* is a data item that specifies the behavior of the FPM application or driver (its network variables and functional blocks). Configuration properties are used for configuration data such as set points, alarm thresholds, or calibration factors. The configuration properties in an FPM can be set by a network management tool such as the SmartServer or the LonMaker tool.

You must declare all the mandatory configuration properties in the UFPT you created for your FPM. You may declare none to all of the optional configuration properties in the UFPT.

### SYNTAX

The syntax used for configuration property declarations is similar to that used for network variable declarations except that the direction modifier is always **input**, and it includes a **config\_prop** or **cp** keyword (you can use either keyword ) that follows the **type** declaration.

```
network input type cp name;  
network input type config_prop name;
```

The **network** keyword declares a configuration property of a specific *type* with a specific *identifier*.

The **input** keyword specifies that the configuration property is an input data point.

The **type** property corresponds to the standard or user-defined network variable type (SCPT or UCPT) used by the configuration property. The specified data type must match the one defined for the referenced configuration property in the UFPT.

The **config\_prop** or **cp** keyword declares the data type as a configuration property.

The **identifier** property is a reference to the configuration property in the UFPT. The name specified in this property is the one that will be used by network tools such as the SmartServer and the LonMaker tool for the referenced configuration property. The maximum length of the identifier is 16 characters.

## EXAMPLES

The follow examples demonstrate how to create configuration properties in your model file:

```
network input SCPTmaxSendTime cp nciHeartbeat;
network input SCPTmaxRcvTime cp nciOffline;
network input SCPTminSendTime cp nciThrottle;
network input UCPTHysteresis cp nciHysteresis;
```

By convention, configuration property names have an *nci* prefix.

For more information on declaring configuration properties, see Chapter 4 of the *Neuron C Programmer's Guide*.

## Declaring Functional Blocks

A *functional block* is a collection of network variables and configuration properties that are used together to perform one task. These network variables and configuration properties are called the *functional block members*. Each functional block implements an instance of a functional profile.

A *functional profile* is used to describe common units of functional behavior. Each functional profile defines mandatory and optional network variables and configuration properties. A functional block must implement all of the mandatory network variables and configuration properties defined by the functional profile, and it may implement any or all of the optional network variables and configuration properties defined by the functional profile.

## SYNTAX

You can use the following syntax to declare functional blocks.

```
fblock FPT-identifier {fblock-member-list} identifier
[external name] [{fb-property-list}];
```

The **fblock** keyword declares a functional block for the **FPT-identifier** and **identifier** properties.

The **FPT-identifier** property specifies the name of the UPFT implemented by the functional block.

The **fblock-member-list** property lists each network variable reference declared in the model file. Every mandatory network variable member in a UFPT must be implemented by network variable reference in the model file. Each network variable in the model file can implement only network variable member, and can only be associated with one functional block in the model file. You can list a network variable reference and implement its corresponding network variable member in the UFPT using the following syntax:

```
nv-reference implements nv-member-name
```

The **identifier** property is the name the declared functional block. This name is not exposed to the SmartServer so you can enter any string in this property.

The **external\_name** keyword declares a functional block name that is exposed to the SmartServer. You can specify the **external\_name** keyword and then a descriptive, readable name for the functional block. If you do not specify a functional block name with



**external\_name** property, the SmartServer uses the FPT key of the UFPT for the functional block name. The FPT key is a unique ID (20000 or higher) defined for the UFPT within a resource file set.

The **fb-property-list** is used to implement the configuration properties declared in the model file that apply to the functional block. You can implement one or more functional block configuration properties using the following syntax:

```
fb_properties
    {cp-reference , cp-reference , cp-reference
    };
```

For more information on declaring functional blocks, see Chapter 5 of the *Neuron C Programmer's Guide*.

### *Using Include Directives*

If a data point declared in your UFPT has a data type that references a system include file, you need to insert an include directives at the beginning of your model file that in order for it to access that source file. System include files are installed by the NodeBuilder Development Tool, and they are stored in the LonWorks\NeuronC\Include directory on your computer by default. To insert an include directive in your model file, you can use the bracketed form:

```
#include <filename.ext> //bracketed form
```

For example, if you declare a **SNVT\_temp\_f** data point in your model file, which has a float data type, you need to insert the following include directives at the beginning of your model file:

```
#include <float.h>
```

### *Example Model Files*

The following examples demonstrate how to create model files that instantiate a single functional block; multiple functional blocks based on the same UFPT; and multiple functional blocks with unique UFPTs.

### **Single Functional Block**

The following example demonstrates a model file that declares all the mandatory network variable and configuration property members in the UFPT, which consists of three input network variables, two output network variables, and four functional block configuration properties (one of which is a user-defined type). The example model file then declares a functional block that does the following: lists network variable member implementations of all the declared input and output network variables, declares an external functional block name to be used by the SmartServer, and implements the declared configuration properties.

```
#include <float.h>

network input SNVT_hvac_mode nviHVACMode;
network input SNVT_temp_f nviSetPoint;
network input SNVT_temp_f nviTemp;

network output SNVT_switch nvoAC_OnOff;
network output SNVT_switch nvoFurnace_OnOff;

network input SCPTmaxSendTime cp nciHeartbeat;
network input SCPTmaxRcvTime cp nciOffline;
network input SCPTminSendTime cp nciThrottle;
network input UCPTHysteresis cp nciHysteresis;

fblock UFPTHVACController {
    nviHVACMode implements nviHVACMode;
```

```

    nviSetPoint implements nviSetPoint;
    nviTemp implements nviTemp;
    nvoAC_OnOff implements nvoAirConditioner_OnOff;
    nvoFurnace_OnOff implements nvoFurnace_OnOff;
} fbHVACFunction external_name ("HVAC Function")

fb_properties {
    nciHeartbeat,nciOffline,nciThrottle,nciHysteresis
};

```

## Multiple Functional Blocks with the Same UFPT

The following example demonstrates how to use a functional block array in a model file to create two functional blocks that are instances of the same UFPT. You could then write a single FPM application based on the UFPT. When you deploy the FPM application on the SmartServer and select the XIF generated from this model file, the internal FPM device will include two functional blocks that are separate instances of the same FPM application.

```

#define NUM_SWITCH_ENCODERS 2

network input SNVT_switch nviACSwitch[NUM_SWITCH_ENCODERS];
network input SNVT_switch nviFurnaceSw[NUM_SWITCH_ENCODERS];
network output SNVT_hvac_mode nvoHVACMode[NUM_SWITCH_ENCODERS];

fbblock UFPTSwitchEncoder {
    nviACSwitch[0] implements nviACSwitch;
    nviFurnaceSw[0] implements nviFurnaceSwitch;
    nvoHVACMode[0] implements nvoHVACMode;
} fbSwitchEncoder[NUM_SWITCH_ENCODERS] external_name
("Digital Encoder");

```

## Multiple Functional Blocks with Unique UFPTs

The following example demonstrates a model file that creates two functional blocks that are instances of two different UFPTs. You could then write separate FPM applications for the UFPTs instantiated by the model file. When you deploy the FPM application on the SmartServer and select the XIF generated from this model file, the internal FPM device will include two functional blocks that are instances of their respective FPM applications.

```

//////////first FB instantiated//////////

#include <float.h>

network input SNVT_hvac_mode nviHVACMode;
network input SNVT_temp_f nviSetPoint;
network input SNVT_temp_f nviTemp;

network output SNVT_switch nvoAC_OnOff;
network output SNVT_switch nvoFurnace_OnOff;

network input SCPTmaxSendTime cp nciHeartbeat;
network input SCPTmaxRcvTime cp nciOffline;
network input SCPTminSendTime cp nciThrottle;
network input UCPTHysteresis cp nciHysteresis;

fbblock UFPTHVACController {
    nviHVACMode implements nviHVACMode;
    nviSetPoint implements nviSetPoint;
}

```

```

    nviTemp implements nviTemp;
    nvoAC_OnOff implements nvoAirConditioner_OnOff;
    nvoFurnace_OnOff implements nvoFurnace_OnOff;
} fbHVACFunction external_name ("HVAC Function")

fb_properties {
    nciHeartbeat,nciOffline,nciThrottle,nciHysteresis
};

//////////second FB instantiated//////////

network input SNVT_switch nviACSwitch;
network input SNVT_switch nviFurnaceSw;
network output SNVT_hvac_mode nvoHVACMode;

fbblock UFPTSwitchEncoder {
nviACSwitch implements nviACSwitch;
nviFurnaceSw implements nviFurnaceSwitch;
nvoHVACMode implements nvoHVACMode;

} fbSwitchEncoder external_name ("Digital Encoder");

```

### Multiple Functional Blocks with Multiple UFPTs

The following example demonstrates how to use multiple functional block arrays in a model file to create multiple sets of functional blocks that are instances of their respective UFPTs. You could then write separate FPM applications for the UFPTs instantiated by the model file. When you deploy the FPM applications on the SmartServer and select the XIF generated from this model file, the internal FPM device will include arrays of the functional blocks that are separate instances of their respective FPM applications.

```

#define NUM_HVAC_FBs 2
#define NUM_SWITCH_ENCODER_FBs 2

#include <float.h>

//////////first FB array instantiated//////////

network input SNVT_hvac_mode nviHVACMode[NUM_HVAC_FBs];
network input SNVT_temp_f nviSetPoint[NUM_HVAC_FBs];
network input SNVT_temp_f nviTemp[NUM_HVAC_FBs];

network output SNVT_switch nvoAC_OnOff[NUM_HVAC_FBs];
network output SNVT_switch nvoFurnace_OnOff[NUM_HVAC_FBs];

network input SCPTmaxSendTime cp nciHeartbeat[NUM_HVAC_FBs];
network input SCPTmaxRcvTime cp nciOffline[NUM_HVAC_FBs];
network input SCPTminSendTime cp nciThrottle[NUM_HVAC_FBs];
network input UCPTHysteresis cp nciHysteresis[NUM_HVAC_FBs];

fbblock UFPTHVACController {
    nviHVACMode[0] implements nviHVACMode;
    nviSetPoint[0] implements nviSetPoint;
    nviTemp[0] implements nviTemp;
    nvoAC_OnOff[0] implements nvoAirConditioner_OnOff;
    nvoFurnace_OnOff[0] implements nvoFurnace_OnOff;
} fbHVACFunction[NUM_HVAC_FBs] external_name ("HVAC Function")
fb_properties {

```

```

    nciHeartbeat[0],nciOffline[0],nciThrottle[0],nciHysteresis[0]
};

//////////second FB array instantiated//////////

network input SNVT_switch nviACSwitch[NUM_SWITCH_ENCODER_FBs];
network input SNVT_switch nviFurnaceSw[NUM_SWITCH_ENCODER_FBs];
network output SNVT_hvac_mode nvoHVACMode[NUM_SWITCH_ENCODER_FBs];

fblock UFPTSwitchEncoder {
nviACSwitch[0] implements nviACSwitch;
nviFurnaceSw[0] implements nviFurnaceSwitch;
nvoHVACMode[0]implements nvoHVACMode;

} fbSwitchEncoder[NUM_SWITCH_ENCODER_FBs] external_name ("Digital
Encoder");

```

### Multiple Functional Blocks with Multiple UFPTs and Same Data Point Names

The following example demonstrates how to use multiple functional block arrays in a model file to create multiple sets of functional blocks that are instances of their respective UFPTs. In addition, this example handles the scenario in which the data points in a UFPT have the same names. You could then write separate FPM applications for the UFPTs instantiated by the model file. When you deploy the FPM applications on the SmartServer and select the XIF generated from this model file, the internal FPM device will include arrays of the functional blocks that are separate instances of their respective FPM applications.

```

#define NUM_OF_ADD_FB 3
#define NUM_OF_SUB_FB 2

//////////first FB array instantiated//////////

network input SNVT_count in1[(NUM_OF_ADD_FB + NUM_OF_SUB_FB)];
network input SNVT_count in2[(NUM_OF_ADD_FB + NUM_OF_SUB_FB)];
network output SNVT_count out1[(NUM_OF_ADD_FB + NUM_OF_SUB_FB)];

fblock UFPTMathAdd {
in1[0] implements in1;
in2[0] implements in2;
out1[0] implements out1;
} fbMathAddFunction[NUM_OF_ADD_FB] external_name ("FpmAdd");

//////////second FB array instantiated//////////

fblock UFPTMathSubtract {
in1[NUM_OF_ADD_FB] implements in1; //can't use [0] for second FB
in2[NUM_OF_ADD_FB] implements in2; //so use next unused index
out1[NUM_OF_ADD_FB] implements out1;
} fbMathSubtractFunction[NUM_OF_SUB_FB] external_name ("FpmSub");

```

### *Saving your Model File*

When you have finished creating your model, you need to save it as a Neuron C source file (.nc extension) on your computer. The example above is stored in a model file named "HVAC.nc" in a folder named "ModelFile" that has been created under the C:\LonWorks directory. The file path of the source file in the example is therefore C:\LonWorks\ModelFile\HVAC.nc. You can then proceed to the next section, which explains how to convert a model file to a XIF.

In addition, you should create a <YourCompany> folder for your company under the C:\LonWorks\Import folder if one does not already exist. This is where the XIF generated by the i.LON SmartServer 2.0 LonWorks Interface Developer tool will be stored.

---

## Generating a Device Interface (XIF) File

You can convert a model file to a XIF using the i.LON SmartServer 2.0 LonWorks Interface Developer tool. This tool is a command line interface that requires you to type a few simple commands to create the XIF. You just need to open a Command Prompt window and specify the file path of your model file, your company's program ID, and the destination file path where the XIF is to be stored. Once it has been generated, you can copy the XIF (.xif extension) from the destination file path to the root/lonWorks/Import/<YourCompany> folder on the SmartServer flash disk.

To generate a XIF and copy it your SmartServer, follow these steps

1. Verify that the full version of the i.LON SmartServer 2.0 Programming Tools has been installed on your computer. Installing the i.LON SmartServer 2.0 Programming Tools installs the i.LON SmartServer 2.0 LONWORKS Interface Developer tool that you will use to create the XIF. For more information on installing the i.LON SmartServer 2.0 Programming Tools, see Chapter 2.
2. Open a Command Prompt window and then type the following command:

```
libilon --source=<model file path> --pid=<program ID> --  
out=<destination path> --basename=<XIF name >
```

For the example HVAC.nc model file shown in the previous section, you would type the following at the command prompt (you would need to replace the sample program ID with your company's program ID, and you would need to replace the "YourCompany" folder in the C:\LonWorks\Import directory with your company's folder):

```
libilon --source=C:\LonWorks\ModelFile\HVAC.nc --  
pid=9F:FD:3E:00:00:00:04:00 --out=  
C:\LonWorks\Import\YourCompany\HVAC --basename=HVAC
```

This creates device interface files named "HVAC" (.xif and .xfb extensions), and stores them in the C:\LonWorks\Import<YourCompany> folder.

### Notes:

- The syntax used in the previous example demonstrates how to use the long form of the command switches. Most command switches also have a short form that you can use. If you wanted to use the short form of the required command switches used in the previous example, you could type the following:

```
libilon -n=C:\LonWorks\ModelFile\HVAC.nc -  
i=9F:FD:3E:00:00:00:04:00 -o=C:\LonWorks\Import\YourCompany -  
b=HVAC
```

Note that if you use the long form, you must insert a separator character (a space or the equals sign [=]) between the command switch and the argument. If you use short form, the separator character is optional.

See the next section, *Using Long and Short Command Switch Forms*, for more information on using the short and long forms.

- You must separate the command switches with spaces.
3. Copy the XIF (.xif extension) generated in step 2 to the root/lonWorks/Import/<YourCompany> folder on the SmartServer flash disk. Note that you may need to create the <YourCompany> folder before copying the XIF.

## Using Long and Short Command Switch Forms

Most command switches come in long and short forms. The long form consists of the verbose, case-sensitive name of the command, and it must be prefixed with a double dash '- -'. Long command switches require a separator (a single space or the equals sign [=]) between the command switch and its respective argument.

The short form consists of a single, case-sensitive, character that identifies the command, and it must be prefixed with a single forward slash '/' or a single dash '-'. Optionally, short command switches may be separated from their respective arguments with a separator (a single space or the equals sign [=]).

The following table demonstrates the long and short forms of the command switches you will typically use to create XIFs.

Long Form	Short Form
--source <file path>	-n <file path>
--pid=<program ID>	/i=<program ID>
--out=<destination path>	/o<destination path>
--basename=<file    filepath>	/b <file>

**Note:** If no command switches or arguments follow the command name, the tool responds with usage hints and a list of available command switches.

## Other Command Switches

The following section lists the long and short forms of the other switches accepted by the **libilon** command and describes the listed command switches.

Long Form	Short Form	Description
--help	-?	Display usage hint for the command
--file	-@	Include a command file
--define	-D	Define a specified preprocessor symbol (without value)
--defloc		Location of an optional default command file
--include	-I	Add the specified folder to the include search path
--mkscript		Generate command script in specified location
--nodefaults		Disable processing of default command files
--silent		Suppress banner message display
--verbosecomments	-V	Generate verbose comments
--verbose	-v	Run with one of the following verbosity levels: 0(normal); 1(verbose); 2(trace)
--warning		Display specified message type as a warning

## Creating FPMs

This chapter describes how to use the *i.LON SmartServer 2.0 Programming Tool* to create new FPM projects and then write, compile, and debug FPM applications and FPM drivers.

---

## Creating FPMs Overview

You can use the full version of the *i.LON SmartServer 2.0 Programming Tool* to create FPMs. The full version of the *i.LON SmartServer 2.0 Programming Tool* includes all the components needed to manage an FPM project:

- Eclipse Development Kit preconfigured for writing, building, and uploading FPMs.
- FPM template files.
- FPM library. A tool for creating the C structures of user-defined UNVTs is also included.
- C++ compiler.
- CYGWIN environment.

Creating an FPM with the full version of the *i.LON SmartServer 2.0 Programming Tool*, entails creating a new FPM project or opening an existing project, writing the FPM application or FPM driver in C or C++, and then compiling the FPM.

You can create a new FPM project from the user-defined functional profile template (UFPT) that you generated with NodeBuilder Resource Editor and you uploaded to your SmartServer following the steps described in Chapter 3, *Creating FPM Templates*. When you create the new FPM project, you select whether you are creating an FPM application or an FPM driver. An FPM application reads and writes values to the data points declared in it, executes an algorithm upon data point updates, reads data point properties, and controls timers and executes code upon their expiration. An FPM driver provides values for the data points declared in it by reading and writing to the RS-232 and RS-485 ports on the SmartServer. Once the FPM project has been created, you can add any user-defined data types to your FPM that were not declared in the UFPT.

After you create a new FPM project or import an existing FPM project, you can write the FPM application or FPM driver in C or C++. Writing the FPM application or FPM driver essentially requires you to implement four main routines that specify the behavior of your FPM:

`Initialize()`, `Work()`, `Shutdown()`, and `OnTimer()`. The `Initialize()` routine is executed when the FPM is started or enabled; the `Work()` routine is executed when a data point declared in the FPM is updated; the `Shutdown()` routine is executed when the FPM is stopped or disabled; and the `OnTimer()` routine is executed when a timer expires.

You can debug your FPMs using a source level debugger (VxWorks 6.2 - Wind River Workbench 2.4) that you can purchase from Wind River. If you are not using Wind River Workbench to debug your FPMs, you should follow the FPM coding guidelines described in this chapter so that you can debug your code more easily. For example, you should frequently insert `printf()` statements in your code so that you can view the status of your FPM using the console port of the SmartServer. For more information on ordering “WindRiver Platform for Industrial Services V3.2 for MIPS32 Processors”, which includes Wind River Workbench, contact Wind River sales at [www.windriver.com/company/contact/index.html](http://www.windriver.com/company/contact/index.html).

Once you finish writing the FPM, you can compile it. If your code has any errors or warnings, they will be displayed in the **Problems** view at the bottom of the document window. You can click on the errors and warnings listed in this view to debug your FPM. You can also check the **Console** view (located to the right of the **Problems** view) to see if there is more detailed information available for a given compiler error or warning.

**Notes:** To use the full version of the *i.LON SmartServer 2.0 Programming Tool*, you must order and install the *i.LON SmartServer 2.0 Programming Tools DVD*. To order the *i.LON SmartServer 2.0 Programming Tools DVD* (Echelon part number 72111-409), contact your Echelon sales representative. If you have a demo version of the *i.LON SmartServer 2.0 Programming Tool*, you can write FPMs, but you cannot compile and upload them.

If you have already created an FPM project and you want to modify your FPM application or FPM driver, you can import your FPM project into the current workspace following the steps described in the *Importing FPM Projects* section in Chapter 2. After you import your FPM project, you can proceed to the *Writing an FPM Application* or *Writing an FPM Driver* section.



---

## Creating New FPM Projects

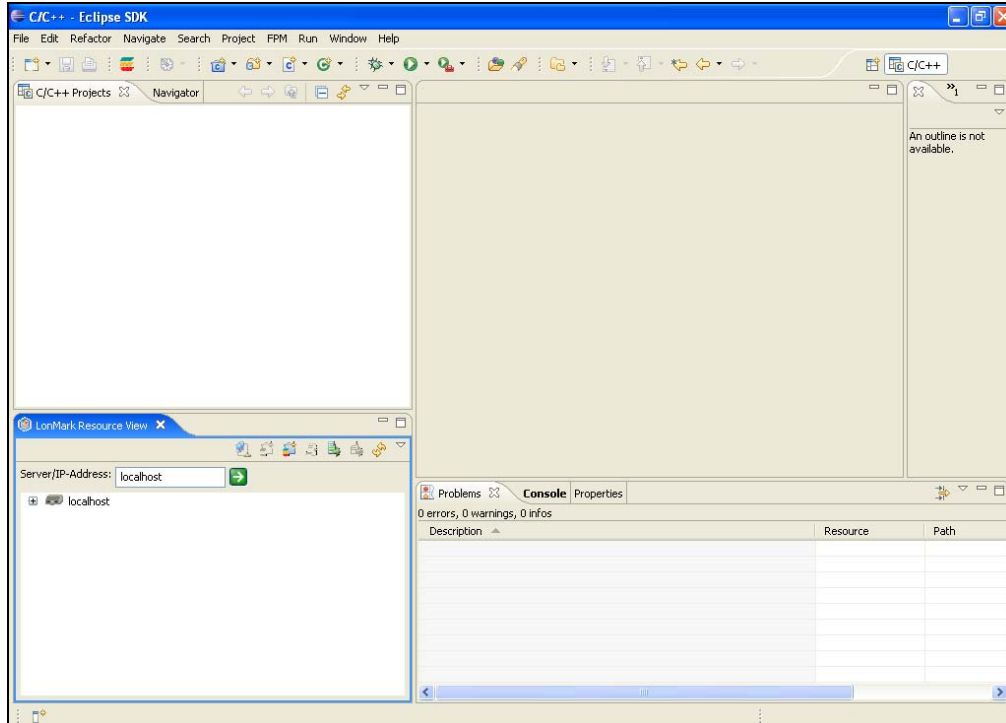
You can create a new FPM project using the *i.LON SmartServer 2.0 Programming Tool*. To create a new FPM project, you do the following:

1. View the resource files on your SmartServer.
2. Create a new FPM application or FPM driver from the resource file set you added to the SmartServer flash disk.
3. Declare the data points to which the FPM will read and write.

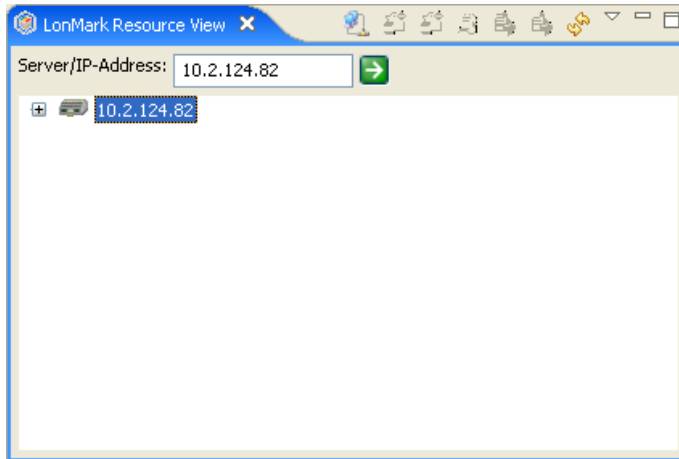
### Viewing the Resource Files on a SmartServer

To view the resource files on a SmartServer, follow these steps:

1. Start the *i.LON SmartServer 2.0 Programming Tool*. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0 Programming Tool* opens.
2. Locate the **LonMark Resource View** at the bottom left-hand corner of the document window. A SmartServer icon named **localhost** appears in this view.



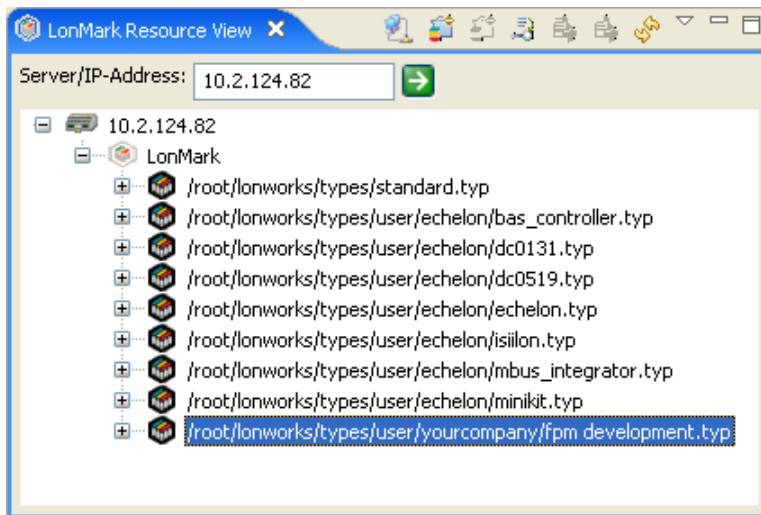
3. In the **Server/IP-Address** box in the **LonMark Resource View**, enter the hostname or IP address of your SmartServer and then click the Go button to the right. The hostname or IP address appears next to the SmartServer icon.



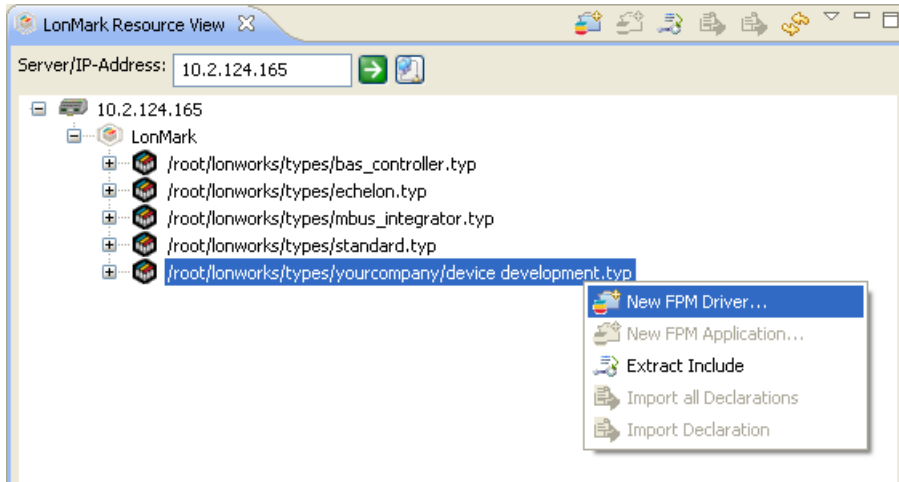
### *Creating an FPM*

To create an FPM application or an FPM driver, follow these steps:

1. Expand the SmartServer icon, and then expand the **LonMark** folder. The resource files in the root/LonWorks/types folder on your SmartServer flash disk are shown.

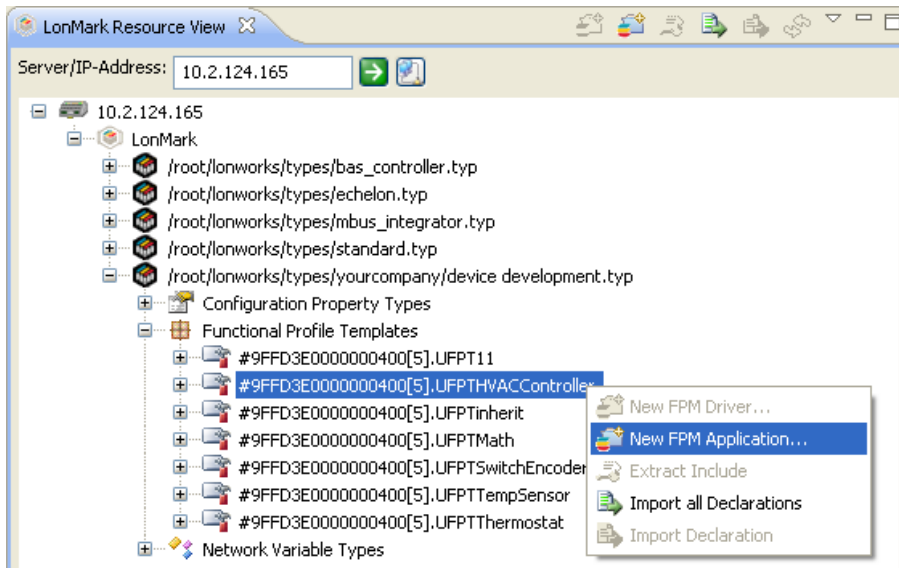


2. To create an FPM driver, right-click your company's FPM resource file set, and then click **New FPM Driver** on the shortcut menu. To create an FPM application, skip to step 3.



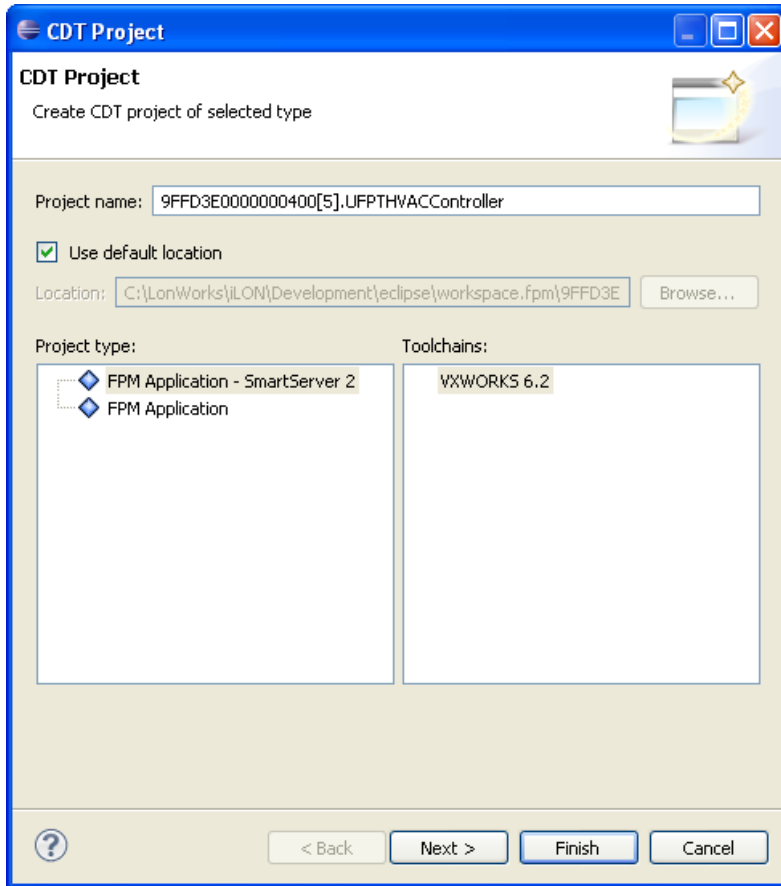
Alternatively, you can click the resource file set and then click the Create FPM icon (🧩) at the top of the **LonMark Resource View**.

3. To create an FPM application, expand your company's FPM resource file set, expand the **Functional Profile Templates** folder, right-click the `<company program ID>.UFPT<FPT Name>`, and then click **New FPM Application** on the shortcut menu.

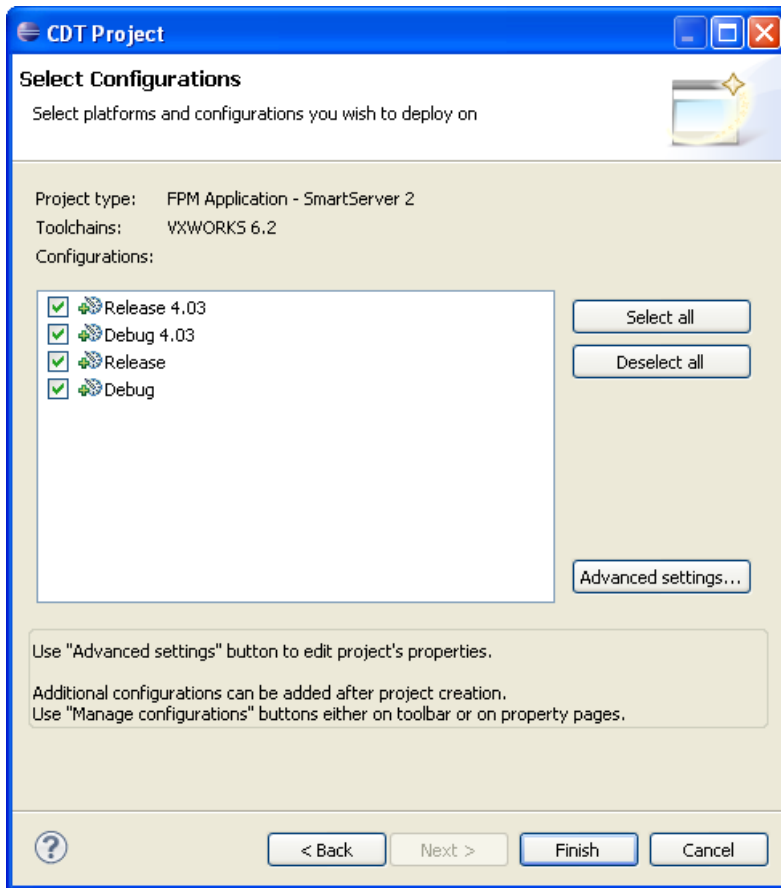


Alternatively, you can click the UFPT and then click the Create FPM icon (🧩) at the top of the **LonMark Resource View**.

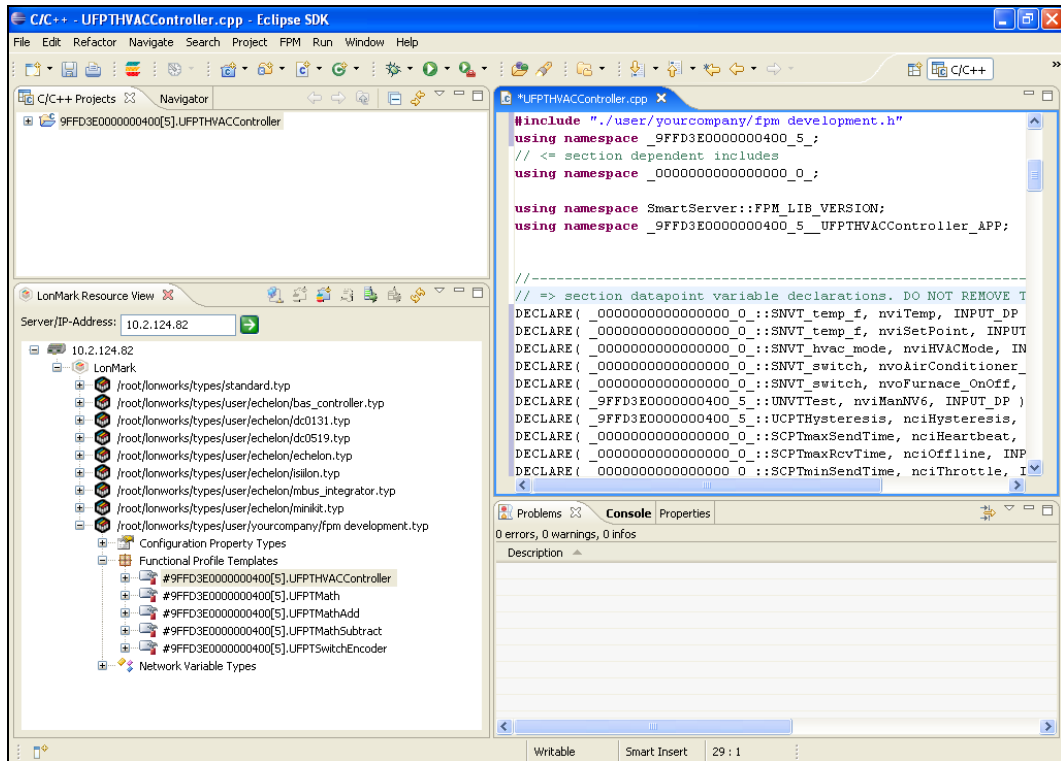
4. The **CDT Project** dialog opens.



5. Optionally, you can specify the project name and the location where the FPM project is to be stored on your computer.
  - a. The default project name is *<company program ID>.UFPT<FPT Name>*. You can accept the default name, which is recommended, or you can enter a different project name. If you enter a different project name, it should adhere to the following naming convention: *<company program ID>. <project>*. This is because your company program ID serves as a namespace that uniquely identifies your FPM and prevents naming collisions with FPMs from different FPM vendors.
  - b. The default location where FPM projects are stored on your computer is `LonWorks\iLON\Development\eclipse\workspace.fpm`. You can accept the default location, or you can specify a different location. When you create a new FPM project, an FPM project folder with the specified project name is added to this directory. The FPM project folder contains the source code (**.cpp** extension), utilities (**utils.cpp** extension), and header (**.h** extension) files for the FPM application.
6. If you will be deploying this FPM on a SmartServer 2.0, accept the default **Project Type**, which is **FPM Application – SmartServer 2**. If you will be deploying this FPM on a SmartServer 1.0, click **FPM Application**.
7. Optionally, you can click **Next** to remove some of the build configurations available for your FPM, or you can click **Finish** to create your FPM project. If you click **Next**, the Select Configurations window opens.



8. Clear the check boxes for any of the configuration you do not plan on deploying. For example, if you are not deploying the FPM on a SmartServer 1.0, you can clear the **Release** check box. If you do not plan on debugging your FPM with WindRiver Workbench, you can clear the **Debug 4.03** and **Debug** check boxes. Click **Finish**.
9. A new FPM project folder with the name specified in step 5 is added to the **C/C++ Projects** view, and the source file view opens to the right of the **C/C++ Projects** view.



10. In the Data Point Variable Declarations section, located just below the namespace declaration, you can observe that `DECLARE` statements have automatically been added to the source file for each data point defined in the UFPT. The data points automatically declared includes standard and user-defined types.

**Note:** If your UFPT includes any user-defined types, your company's header files are automatically added to the `C:\LonWorks\iLON\Development\include` folder located under the FPM project's **Include** folder, and an `#include` directive for your company's header file is automatically inserted in your source file

If you modify the UFPT used by your FPM, you can update the data point declarations as described in the next section, *Declaring Data Points*. Otherwise, you can skip to the *Writing an FPM Application* or *Writing an FPM Driver* section depending on the type of FPM you are creating.

11. If you are creating an FPM driver, expand your company's FPM resource file set, and then manually import the desired network variable declarations into your FPM driver. You can manually import all the network variable declarations in a UFPT, or can manually import individual network variable declarations in a UFPT.
- To manually import all the network variable declarations, see *Manually Importing All Data Point Declarations* in the next section.
  - To manually import individual network variable declarations, see *Manually Importing Individual Data Point Declarations* in the next section.

### Updating Data Point Declarations

If you add new network variable and configuration property members to the UFPT used by your FPM or you modify any of the existing members, you can add the new data points or update the existing data points in the source file (**.cpp** extension).

To update the data points declarations in your source file, you need to copy your company's updated resource file set to the `root/LonWorks /types/User/<Your Company>` folder on the SmartServer flash

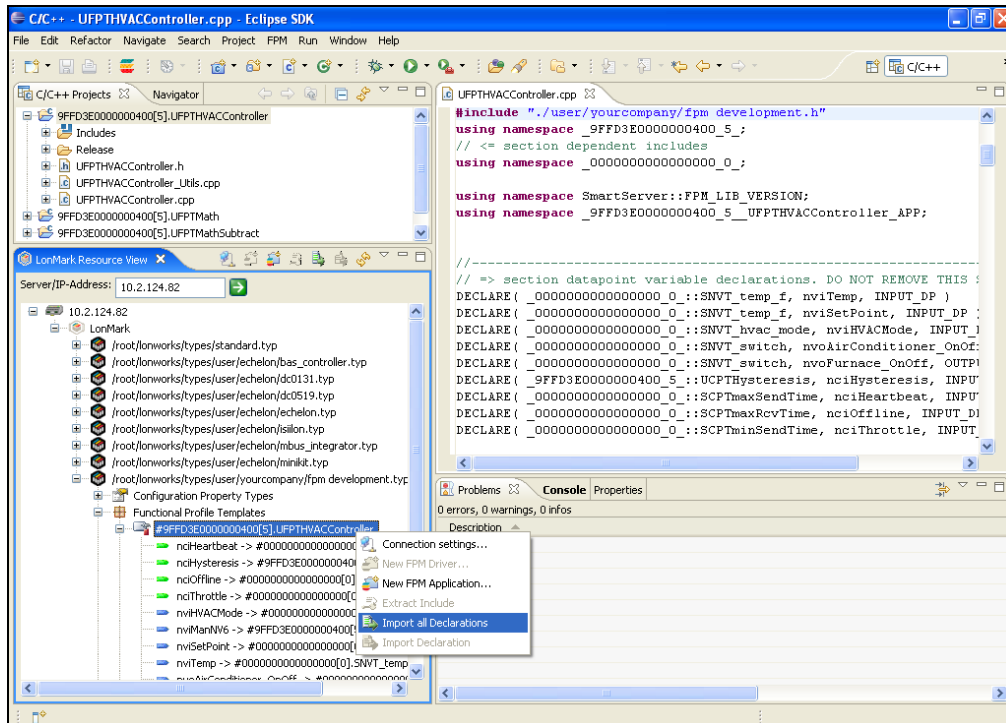
disk. See Chapter 3 for more information on generating an updated resource file set and uploading the updated resource files to the SmartServer.

After you generate and copy your updated resource files set to the SmartServer, you can manually import one to all of the data point declarations.

## Manually Importing All Data Point Declarations

In the source file (.cpp extension) of your FPM, you can add new data points that has been created in the UFPT, and you can update all the existing data points that have been modified in the UFPT. To do this, follow these steps:

1. In the **LonMark Resource View**, right-click the UFPT from which the FPM project was created, and then click **Import All Declarations** on the shortcut menu. Alternatively, you can click the UFPT and then click the Import Declare() for All Data Points icon (📄) at the top of the **LonMark Resource View**.

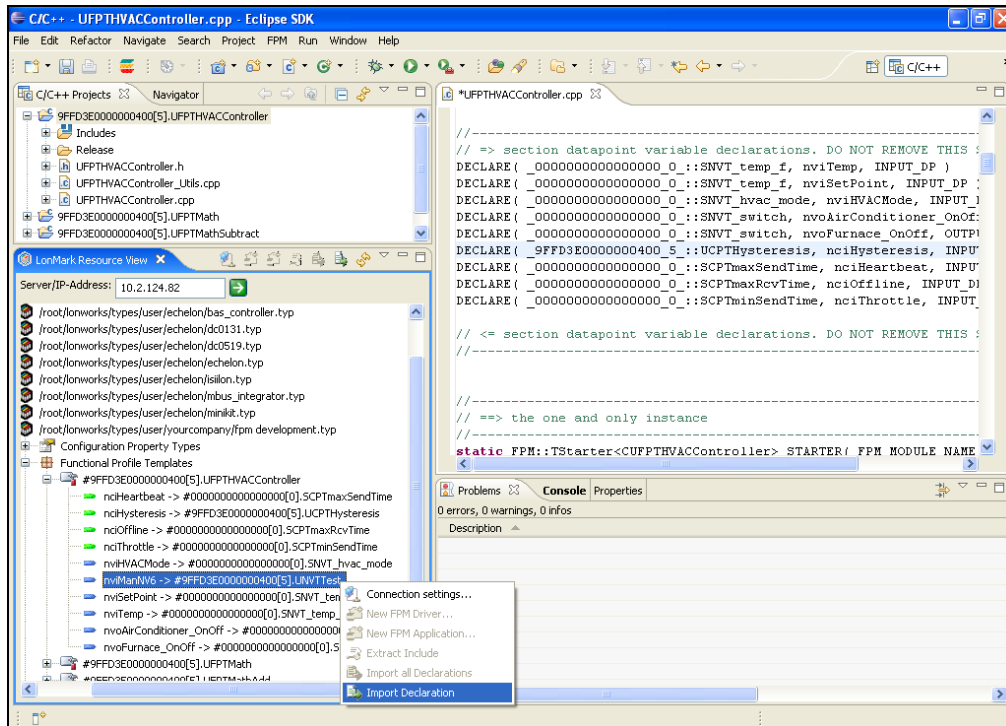


**Note:** Verify that you are currently working in the source file (.cpp extension) before importing the data point declarations (the tab of the current code view is highlighted blue or white depending on whether it has focus). If you add the data point declarations to a different file, your FPM will not function and the file in which the data point declarations were imported may also not function.

2. Updated DECLARE statements for each data point defined in the UFPT are added to the data point declaration section just below the namespace declaration.







3. An updated DECLARE statement for the selected data point is added to the data point declaration section just below the namespace declaration.
4. Repeat steps 2–3 to add or update additional data points in the UFPT.

## Using UFPT Local Variables

Creating an FPM application or driver with the New FPM Project wizard in the *i*.LON SmartServer 2.0 Programming Tool generates a class that inherits from `CFPM_App`. This class provides the implementation for an FPM functional block. At runtime, only one instance of this class will be created for each unique FPM. When you add multiple FPM devices on the SmartServer that use that same unique FPM, multiple functional block instances of the FPM are created. When a `Work()` routine in the FPM is called, the FPM framework provides data point values that are applicable to their respective functional block instances.

If you want to use additional data point variables that also apply to specific functional block instances, you can use the `DECLARE_FB_INSTANCE_LOCAL()` macro. For example, you can declare a UFPT local variable that stores how often the `Work()` routine has been called by specific functional block instance or you can declare a UFPT local variable that stores the file name of a functional block instance.

Consider a scenario in which there is one internal FPM Math device that has two functional block instances of the Math UFPT (Add1 and Add2). The Math FPM application contains three data points (`in1`, `in2`, and `out1`), and it has one local variable (`callCount`) that is incremented when the `Work()` routine is called.

- On the Net/LON/MathFPM/Add1 functional block, `in1` is set to 20, and then `in2` is set to 5. This results in the Add1 functional block calling the `Work()` routine twice. The `out1` data point is updated to 20 and then to 25, and the `callCount` local variable is updated to 2.
- On the Net/LON/MathFPM/Add2 functional block, `in1` is set to 10 (the `in2` data point is not updated). This results in the Add2 functional block calling the `Work()` routine once. The `out1` data point is updated to 10 and the `callCount` local variable is updated to 1.

## SYNTAX

```
DECLARE_FB_INSTANCE_LOCAL(dataType, variableName)
```

## EXAMPLE

```
// <= section datapoint variable declarations  
DECLARE_FB_INSTANCE_LOCAL(int, callCount);
```

---

## Writing an FPM Application

You can write an FPM application using the *i*.LON SmartServer 2.0 Programming Tool. An FPM application reads and writes to the data points declared in it, reads data point properties, executes code upon data point updates, and controls timers and executes code upon their expiration. To create an FPM application, you specify the logic to be executed on the data points in the following four routines: `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()`:

- The `Initialize()` routine is executed when the FPM application is started or enabled. In the `Initialize()` routine, you can set the initial values for data points, and you can start timers using the `Start()` method of the `CFPM_Timer` class or the `START_TIMER()` macro.
- The `Work()` routine is executed when one or more data points declared in the FPM application are updated. In the `Work()` routine, you write an algorithm that is executed when a specified data point is updated. In the algorithm, you can read data point properties and write values to the data points. In addition, you can start and stop timers.
- The `OnTimer()` routine handles timer expiration events. You can use this routine in conjunction with the `Start()` methods or the `START_TIMER()` macros called in the `Initialize()` routine. You can create an algorithm in this routine that read data point properties and writes values to the data points upon the expiration of a timer. In addition, you can start and stop timers.
- The `Shutdown()` routine is executed when the FPM application is stopped or disabled as a result of a reboot. In the `Shutdown()` routine, you stop timers and perform any required cleanup.

The *i*.LON SmartServer 2.0 Programming Tool works with raw data point values. You must remember to use the appropriate scale factors to convert raw data point values to scaled values. You can go to [types.lonmark.org/index.html](http://types.lonmark.org/index.html) to check the scale factors uses for the SCPTs and UCPTs in the current LONMARK standard resource file.

### *The Writing the FPM Application Initialize() Routine*

The `Initialize()` routine is executed when the FPM application is started or enabled. You can use this routine to write initial values to the data points declared in your FPM application. In addition, you can use this routine to start timers, which you can use in an FPM application to implement tasks that need to be performed regularly such as checking the status of data points. For more information on writing values to data points, see the next section, *Writing the FPM Application Work() Routine*.

The following code demonstrates how to set initial data point values and start timers in the `Initialize()` routine:

```
DECLARE(_0000000000000000_0_::SNVT_temp_f, nviSetPoint,  
        INPUT_DP)  
  
CFPM_Timer m_oTimer1; //declared in header file  
CFPM_Timer m_oTimer2; //declared in header file  
CFPM_Timer m_oTimer3; //declared in header file  
  
void CUFPTHVACController::Initialize()  
{
```

```

//set initial data point values
nviSetPoint = 0;
nviTemp = 0;
nciHystereis = -17.77778;

//start timers

m_oTimer1.Start(FPM_TF_REPEAT, 2000);
m_oTimer2.Start(FPM_TF_ONETIME, 3000);
START_TIMER(m_oTimer3, FPM_TF_REPEAT, 2000, OnMyTimer3);

// to do: create OnMyTimer3()routine to handle m_oTimer3
}

```

**Note:** Initialized input data point values are not propagated to output data points when the `Initialize()` routine executes. Input data point values are only propagated to output data points when the `Work()` routine executes as a result of an input data point value changing.

## Declaring and Initializing Timers

To use a timer in your FPM application, you must first declare it as a member of the `CFPM_Timer` application class in the header file (`.h` extension) of your FPM application and then initialize it in the source file (`.cpp` extension). To declare and initialize a timer, follow these steps:

1. Open the header file. To do this, either double-click the header file (`.h` extension) in the **C/C++ Projects** view or right-click it and then click **Open** on the shortcut menu. The header file view opens to the right of the source file view.
2. Scroll to the “Mandatory Application Members” section in the header file, and then declare the timer using the following syntax:

```

CFPM_Timer m_oTimer1; //declare a timer
CFPM_Timer m_oTimer2; //declare a timer
CFPM_Timer m_oTimer3; //declare a timer

```

3. Click the tab for your source file (`.cpp` extension), scroll to the “Constructor/Deconstructor” section, and then initialize the timer using the following syntax:

```

, m_oTimer1(this) //initialize timer
, m_oTimer2(this) //initialize timer
, m_oTimer3(this) //initialize timer

```

## Starting Timers

You can start timers using the standard `Start()` method of the `CFPM_Timer` class or the user-defined `START_TIMER()` macro. Note that when you start timers, you should set the timer interval to a minimum of 100ms.

### Using the Start() Method

The `Start()` method of the `CFPM_Timer` class is the standard approach for starting timers. It causes the FPM application to call the `OnTimer()` routine, which handles the timer expiration event. The `Start()` method has the following syntax:

```
void Start(FPM_TimerFlags_t eMode, uint_t nTimeoutMillis);
```

The `eMode` parameter specifies the type of the timer. You can enter `FPM_TF_REPEAT` for a repeating timer, or you can enter `FPM_TF_ONETIME` for a timer that is used just once.

The `nTimeoutMillis` parameter specifies the timer interval in milliseconds. You should set this parameter to a minimum of 100ms.

The following code demonstrates how to create repeating and one-time timers using the `Start()` method:

```
CFPM_Timer m_oTimer1; // declared in header file
CFPM_Timer m_oTimer2; // declared in header file
, m_oTimer1(this)      // initialized in source file
, m_oTimer2(this)      // initialized in source file
...
m_oTimer1.Start(FPM_TF_REPEAT, 2000);
m_oTimer2.Start(FPM_TF_ONETIME, 3000);
```

### Using the `START_TIMER()` Macro

The `START_TIMER()` macro is an alternative approach to creating and starting timers. It causes the FPM application to call a user-defined timer handler method, which must be declared in the header file (`.h` extension) of your FPM application. Note that you can declare and initialize an unlimited number of timers and create an unlimited number user-defined timer handler methods for them. The `START_TIMER()` method has the following syntax:

```
START_TIMER(timeVar, mode, timeoutMillis, funcName)
```

The `timeVar` parameter specifies the name of the timer to be started.

The `mode` parameter specifies the type of the timer. You can enter `FPM_TF_REPEAT` for a repeating timer, or you can enter `FPM_TF_ONETIME` for a timer that is used just once.

The `nTimeoutMillis` parameter specifies the timer interval in milliseconds. You should set this parameter to a minimum of 100ms.

The `funcName` parameter specifies the name of the user-defined timer handler method that is called when this expires.

Timers started with the `START_TIMER()` macro must be handled in your source file (`.cpp` extension) with a user-defined timer handler method that has the following signature:

```
void <funcName>()
```

You must declare your user-defined timer handler method in the “Implements the User Functionality” section of the header file (`.h` extension) of your FPM application.

The following example demonstrates a `START_TIMER()` macro that starts a timer that repeats every 3 seconds and is handled by the `OnMyTimer3()` user-defined timer handler method.

```
CFPM_Timer m_oTimer3; // declared in header file
, m_oTimer3(this)      // initialized in source file
...
START_TIMER(m_oTimer3, FPM_TF_REPEAT, 3000, OnMyTimer3);
...
void OnMyTimer3();      // declared in header file
```

See the *Programmer's Reference* in Appendix A for more information on starting timers in the `Initialize()` routine and using timer handler methods.

### *Writing the FPM Application `Work()` Routine*

The `Work()` routine is executed when one or more data points declared in the FPM application are updated. In the `Work()` routine, you write one or more IF-THEN(-ELSE) statements that use the `Changed()` method to evaluate whether the data points in the FPM application have been updated and execute an algorithm if they have been updated.

In the algorithm, you can directly read and write values to the data points declared in your FPM application without using any additional methods. In addition, you can read the statuses, names, and times of last update of the data points using a collection of data point property methods, and you can start and stop timers. For more information on starting timers, see the previous section, *Writing the FPM Application Initialize() Routine*.

The following code demonstrates how you can create a `Work()` routine in your FPM application. In this example, the `Work()` routine first checks whether the data points declared in the FPM application have been updated. If one of the data points has been updated, the `Work()` routine gets the statuses of the data points, checks whether the data points are in normal condition, and then reads and writes values to them if they are in normal condition.

```
void CUFPHVACController::Work()
{
    SNVT_switch tempAirConditioner;

    if ((Changed( nviSetPoint ) || Changed( nviTemp )) &&
        (nviHVACMode == hvac_t::HVAC_COOL))
    {
        nviSetPoint_status =
            nviSetPoint.GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus);
        printf ("nviSetPoint_status = %d", nviSetPoint_status);

        nviTemp_status =
            nviTemp.GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus);
        printf ("nviTemp_status = %d", nviTemp_status);

        if (nviSetPoint_status == FPM::Dp::AL_NO_CONDITION) &&
            (nviTemp_status == FPM::Dp::AL_NO_CONDITION)
        {
            if (nviTemp > (nviSetPoint + nciHysteresis))
            {
                nvoAirConditioner_OnOff->value = 200;
                nvoAirConditioner_OnOff->state = 1;
                PROPAGATE (nvoAirConditioner_OnOff);
                printf ("Temp = %f, SetPoint=%f, AC is
                    ON \n", *nviTemp, *nviSetPoint);
            }
        }
    }
}
```

### Checking for Data Point Value Updates

You can use the `Changed()` method in the `Work()` routine to determine whether the value of a data point has changed. This method takes a data point declared in the FPM application. If the value of the data point has changed, it returns `TRUE`; otherwise, it returns `FALSE`. The following example demonstrates how you can use the `Changed()` method to check whether the values of the data points in your FPM application have changed.

```
if (Changed(x) || Changed(y))
{
    //execute some algorithm if the values of x or y have changed
}
```

**Note:** You can modify the behavior of the `Changed()` method so that it can determine whether any data point property has changed, including value, status, time of last update, and priority. See the next section, *Checking for Data Point Property Updates*, for how to do this.

## Checking for Data Point Property Updates

You can call the `NotifyOnAllUpdates()` method in the constructor of your FPM application so that the `Changed()` method checks whether any data point property has been updated, including value, status, time of last update, and priority (by default, the `Changed()` method only checks whether the data point value has changed).

This is ideal if you are using FPMs with external devices that require the same value to be written to a data point to execute some specific device behavior. In this case, when the same value is written to the data point (via polling or the same value being explicitly written to the data point), the `Changed()` method returns `TRUE` because the time of last update property has been updated. Another common use-case is the checking of devices for alarm conditions.

In the `NotifyOnAllUpdates()` method, you pass in a string vector containing the names of the data points for which the `Changed()` method is to check for updates to any of their properties. The following example demonstrates how to use the `NotifyOnAllUpdates()` and the `Changed()` methods to check whether the properties of specific data points in your FPM application have changed.

**Note:** If you do not call the `NotifyOnAllUpdates()` method within the FPM constructor, the `Changed()` method only checks whether the data point value has changed.

```
// FPM constructor
CUFPTHVACController::CUFPTHVACController()
    : CFPM_App(FPM_MODULE_NAME, CFPM_App::eApplication)
{
    vector<string> oDpVarNames;
    oDpVarNames.push_back("nviTemp");
    oDpVarNames.push_back("nvoAC_OnOff");
    NotifyOnAllUpdates(oDpVarNames);
}
.....

// section datapoint variable declarations
DECLARE(_0000000000000000_0::SNVT_temp_f, nviTemp, INPUT_DP)
DECLARE(_0000000000000000_0::SNVT_switch, nvoAC_OnOff, INPUT_DP)

.....

// Work() routine
void CUFPTHVACController::Work()
{
    FPM::Dp::PointStatus nviTemp_status;
    nviTemp_status =
    nviTemp.GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus);
    // check whether nviTemp status has changed
    if (Changed(nviTemp))
    {
        if (nviTemp_status == FPM::Dp::AL_HIGH_LMT_ALM_1)
        {
            nvoAC_OnOff->value = 200;
            nvoAC_OnOff _OnOff->state = 1;
            PROPAGATE(nvoAC_OnOff);
        }
    }
}
```

## Reading Data Point Properties

You can read the name, alias name, time of last update, and status of each data point declared in the FPM application in the `Initialize()`, `Work()`, and `OnTimer()` routines. To read these data point properties, you use a collection of `get` property methods belonging to each data point. The methods that you can call to read the data point properties are as follows:

Data Point Property	Get Property Method
Name	<code>const char* GetDpPropertyAsString(FPM::Dp::cfgUCPTname)</code>
Alias name	<code>const char* GetDpPropertyAsString(FPM::Dp::cfgUCPTAliasName)</code>
Time of Last Update	<code>timespec GetDpPropertyAsTimeSpec(FPM::Dp::dataUCPTlastUpdate)</code>
Status	<code>FPM::Dp::PointStatus GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus)</code>

The following code demonstrates how to read data point properties using these methods:

```
void CUFPHVACController::Work()
{
    const char* nviSetPoint_name;
    const char* nviSetPoint_AliasName;
    timespec nviSetPoint_lastUpdateTime;
    FPM::Dp::PointStatus nviTemp_status;

    nviSetPoint_name =
        nviSetPoint.GetDpPropertyAsString(FPM::Dp::cfgUCPTname);

    nviSetPoint_AliasName =
        nviSetPoint.GetDpPropertyAsString(FPM::Dp::cfgUCPTAliasName);

    nviSetPoint_lastUpdateTime =
        nviSetPoint.GetDpPropertyAsTimeSpec(FPM::Dp::dataUCPTlastUpdate);

    nviTemp_status =
        nviTemp.GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus);

    if (nviTemp_status == FPM::Dp::AL_ALM_CONDITION)
    {
        nvoHVACMode = hvac_t::HVAC_COOL;
    }
}
```

## Reading Data Point Values

You can directly read the values of the scalar, structured, and enumerated data points declared in your FPM application in the `Work()` routine. Upon reading the data points, you can execute code based on the current values stored in them.

## Reading Scalar Data Points

You can directly read the value of a scalar data point declared in your FPM by simply referencing its name.

```
void CUFPHVACController::Work()
{
    if (nviTemp > nviSetPoint + (nciHysteresis*1.8+32))
        //execute some code
}
```

```

    }
}

```

## Reading Structured Data Points

To read the value of a structured data point, you first need to store the data point field in a temporary data point variable. Note that you use the `->` operator (element selection through pointer) to reference the fields of a structured data point. The following code demonstrates how to read the fields of a structured data point:

```

void CUFPHVACController::Work()
{
    SNVT_switch tmpSwitch;

    tmpSwitch.state = nvoAirConditioner_OnOff->state;
    tmpSwitch.value = nvoAirConditioner_OnOff->value;

    if (tmpSwitch.state == 0)
    {
        //insert code here
    }

    if (tmpSwitch.value >= 145)
    {
        //insert code here
    }
}

```

## Reading Enumerated Data Points

To read the value of an enumerated data point, you need to use the values of the corresponding enumeration type. The following code demonstrates how to read the following enumeration types:

- An `hvac_t` enumeration in the **standard.h** file in the `LonWorks\iLON\Development\include` folder
- A `pointStatus` enumeration in the **FPM\_variable.h** file in the `LonWorks\iLON\Development\eclipse\plugins\com.echelon.eclipse.ilon100.fpm_0.9.0\compiler\echelon\fpm\include` folder.

```

DECLARE(_0000000000000000_0_::SNVT_temp_f, nviSetPoint, INPUT_DP)
DECLARE(_0000000000000000_0_::SNVT_hvac_mode, nviHVACMode, INPUT_DP)

```

```

.....
void CUFPHVACController::Work()
{
    FPM::Dp::PointStatus nviSetPoint_status = FPM::Dp::AL_NUL;

    if (nviHVACmode == hvac_t::HVAC_HEAT) //HVAC_HEAT
    {
        nviSetPoint_status = (FPM::Dp::PointStatus)
            nviSetPoint.GetDpPropertyAsPointStatus
                (FPM::Dp::dataUCPTstatus);
    }

    if (nviSetPoint_status == FPM::Dp::AL_NO_CONDITION)
    {
        //insert code here
    }
}

```



## Writing Data Point Values

You can write updated values to the scalar and structured data points declared in your FPM application in the `Work()` routine. The updated values are then written back to the data points when the FPM application exits the `Work()` routine. You can directly write an updated value to a scalar data point by simply assigning it a value.

```
nviTemp = 25.28;
nviSetPoint = 22.22;
```

Writing an updated value to a structured data point requires a few additional steps. You can directly write values to the fields of a structured data point and then mark the data point as modified, or you can create temporary data point variables and use them to write values to the fields of your structured data points. The following sections describe how to write values to a structured data point using each of these methods.

### Directly Writing to a Structured Data Point

To directly write a value to a structured data point, you use the `->` operator (element selection through pointer), and you then mark the data point as modified using the `PROPAGATE()` macro. The following code demonstrates how to write to a structured data point using this method:

```
DECLARE(_0000000000000000_0_::SNVT_switch,
        nvoAirConditioner_OnOff, OUTPUT_DP);
...
void CUFPHVACController::Work()
{
    // use variable directly and tell the system that this value /
    // should be written back to the data point
    nvoAirConditioner_OnOff->value = 200;
    nvoAirConditioner_OnOff->state = 1;
    PROPAGATE(nvoAirConditioner_OnOff);
}
```

### Using Temporary Data Point Variables to Write to a Structured Data Point

To use a temporary data point variable to write to a structured data point, you declare a temporary data point in the `Work()` routine, store the desired values in the various fields of the temporary data point, and then assign the declared data point a reference to the temporary data point variable. The following code demonstrates how to write to a structured data point using this method.

```
DECLARE(_0000000000000000_0_::SNVT_switch,
        nvoAirConditioner_OnOff, OUTPUT_DP)
...
void CUFPHVACController::Work()
{
    SNVT_switch tmp;
    tmp.value = 200;
    tmp.state = 1;
    nvoAirConditioner_OnOff = tmp;

    // change detected and written to data point automatically
}
```

See the *Programmer's Reference* in Appendix A for more information on writing values to structured data points.

## Writing to Enumerated Data Points

To write to an enumerated data point, you need to use the values of the corresponding enumeration type. The following code demonstrates how to write to an `hvac_t` enumeration in the `standard.h` file in the `LonWorks\iLON\Development\include` folder:

```
DECLARE(_0000000000000000_0_::SNVT_hvac_mode, nvoHVACMode, OUTPUT_DP)
.....
void CUFPTHVACController::Work()
{
    SNVT_switch ACSwitch;
    SNVT_switch FurnaceSwitch;
    ACSwitch.state = nviACSwitch->state;
    FurnaceSwitch.state = nviFurnaceSwitch->state;

    if ((tmpACSwitch.state == 1) &&
        (tmpFurnaceSwitch.state == 0))
    {
        nvoHVACMode = hvac_t::HVAC_COOL;
        printf ("HVAC MODE = %i \n", *nvoHVACMode);
    }
}
```

## Writing the FPM Application OnTimer() Routine

The `OnTimer()` routine is executed when a timer started with the `Start()` method in the `Initialize()` routine expires. You can use this routine to read the properties of the data points declared in the FPM application. This is useful for implementing tasks that need to be performed regularly such as checking data point status and sending data point updates (heartbeats). For more information on reading data point properties in an FPM application, see the previous section, *Writing the FPM Application Work() Routine*.

Note that if you started more than one timer using the `Start()` method of the `CFPM_Timer` class, you must first identify the timer that expired using the `m_oTimer.Expired()` method.

The following code demonstrates how to create an `OnTimer()` routine that handles the expiration of a single timer started with the `Start()` method of the `CFPM_Timer` class.

```
void CUFPTHVACController::Initialize()
{
    m_oTimer1.Start(FPM_TF_REPEAT, 2000);
}

void CUFPTHVACController::OnTimer()
{
    //check for data point alarm conditions

    if (nviSetPoint_status == FPM::Dp::AL_ALM_CONDITION)
    {
        printf ("SetPoint status = %d", nviSetPoint_status);
        nviSetPoint = 20.28;
    }
    if (nviTemp_status == FPM::Dp::AL_ALM_CONDITION)
    {
        printf ("Temp status = %d", nviTemp_status);
        nvoHVACMode = hvac_t::HVAC_COOL;
    }
}
```

The following code demonstrates how to create an `OnTimer()` routine that handles the expiration of multiple timers started with the `Start()` method of the `CFPM_Timer` class. Observe that the `m_oTimer.Expired()` method is called first to determine which of the two timers started in the `Initialize()` routine expired.

```
void CUFPHTHVACController::Initialize()
{
    m_oTimer1.Start(FPM_TF_REPEAT, 2000);
    m_oTimer2.Start(FPM_TF_ONETIME, 3000);
}

void CUFPHTHVACController::OnTimer()
{
    if (m_oTimer1.Expired())
    {
        // do some task
    }

    if (m_oTimer2.Expired())
    {
        // do some task
    }
}
```

You must create custom `OnMyTimer()` routines for each user-defined timer you started with the `START_TIMER()` macro in the `Initialize()` routine. The following code demonstrates how to create `OnMyTimer()` routines that handle the expiration of their respective timers started with the `START_TIMER()` macro.

```
void CUFPHTHVACController::Initialize()
{
    START_TIMER(m_oTimer3, FPM_TF_REPEAT, 2000, OnMyTimer3);
    START_TIMER(m_oTimer4, FPM_TF_REPEAT, 2000, MyTimerHandler4);
}

void CUFPHTHVACController::OnMyTimer3()
{
    //do some task to handle expiration of the m_oTimer3 timer
}

void CUFPHTHVACController::MyTimerHandler4()
{
    //do some task to handle expiration of the m_oTimer4 timer
}
```

See the *Programmer's Reference* in Appendix A for more information on starting timers and using timer handler methods.

### *Writing the FPM Application Shutdown() Routine*

The `Shutdown()` routine is executed when the FPM application is stopped or disabled. In the `Shutdown()` routine, you stop timers and perform any required cleanup. You can stop a timer using use the `Stop()` and `StopAllTimers()` methods of the `CFPM_Timer` class.

- The `StopTimer()` method causes the system to stop the referenced timer.
- The `StopAllTimers()` method causes the system to stop all timers.

The following code demonstrates how you can stop timers in the `Shutdown()` routine:

```

void CUFPTHVACController::Shutdown()
{
    m_oTimer1.Stop();
    StopAllTimers();
}

```

See the *Programmer's Reference* in Appendix A for more information on stopping timers.

---

## Writing an FPM Driver

You can write an FPM driver using the *iLON SmartServer 2.0 Programming Tool*. An FPM driver creates data points on the SmartServer and provides values for them by reading and writing to the RS-232 and RS-485 ports on the SmartServer. To create an FPM driver, you specify the logic to be executed on the data points in the following four routines: `Initialize()`, `Work()`, `Shutdown()`, and `OnTimer()`:

- The `Initialize()` routine is executed when the FPM driver is started or enabled. In the `Initialize()` routine, you start timers using the `Start()` method of the `CFPM_Timer` class or the `START_TIMER()` macro, open the RS-232 and RS-485 ports on the SmartServer, and write to the properties of the data points declared in the FPM driver.
- The `Work()` routine is executed when one or more data points declared in the FPM driver are updated. In the `Work()` routine, you write one or more IF-THEN(-ELSE) statements that evaluate whether the values of the data points in the FPM have been updated. If the data points have been updated, you initialize communication between your FPM and the RS-232 or RS-485 serial interface and then write to the interface. You can also read the properties of the data points declared in the FPM driver in this routine.
- The `OnTimer()` routine handles timer expiration events. In the `OnTimer()` routine, you initialize communication between your FPM and the RS-232 or RS-485 serial interface when a timer started in the `Initialize()` routine expires. You can then read and write to the RS-232 or RS-485 interface, write updated values to the data point declared in the FPM driver, and read the properties of the data points.
- The `Shutdown()` routine is executed when the FPM driver is stopped or disabled as a result of a reboot. In the `Shutdown()` routine, you stop timers, close the RS-232 and RS-485 ports on the SmartServer, and perform any required cleanup.

**Note:** The `LonWorks\iLON\Development\eclipse\workspace.fpm` directory on your computer includes a sample RS-232 driver that you can view, edit, compile, and deploy on your SmartServer.

### Writing the FPM Driver `Initialize()` Routine

The `Initialize()` routine is executed when the FPM driver is started or enabled. You can use this routine to start timers; open the RS-232 or RS-485 serial interface on the SmartServer; and write to the properties of the data points declared in the FPM driver.

- You can start timers using the `Start()` method of the `CFPM_Timer` class and the `START_TIMER()` macro as described in *Writing the FPM Application `Initialize()` Routine*.
- You can open the RS-232 serial interface using the `rs232_open()` method. You can open the RS-485 serial interface using the `rs485_open()` method.
- You can set the default values, persistent flags, poll rates, and unit strings of the data points declared in the FPM driver using the following collection of data point properties.

```

//write data point default value in raw hex
Line1.SetDpProperty(FPM::Dp::cfgUCPTdefOutput, nValue);

//set whether data point is persistent

```

```

Line1.SetDpProperty(FPM::Dp::cfgUCPTpersist, bValue);
//write data point poll rate in milliseconds
Line1.SetDpProperty(FPM::Dp::cfgUCPTpollRate, nValue);
// write data point unit string
Line1.SetDpProperty(FPM::Dp::cfgUCPTunit, pszValue);

```

The following code demonstrates the methods you can create in the `Initialize()` routine in an FPM driver for the RS-232 interface:

```

CFPM_Timer m_oDisplay_InputTimer; //declare timer in header file
.....

DECLARE(_0000000000000000_0::SNVT_str_asc, Line1, INPUT_DP)
DECLARE(_0000000000000000_0::SNVT_switch, F1_Pressed, OUTPUT_DP)
int RS232_fd = -1;

void CUFPT_Display::Initialize()
{
    // start timer
    m_oDisplay_InputTimer.Start(FPM_TF_REPEAT, 10000);

    // open the RS232 interface
    RS232_fd = rs232_open(RS232_DEFAULT_BAUDRATE);

    //set Line1 DP poll rate
    F1_Pressed.SetDpProperty(FPM::Dp::cfgUCPTpollRate, 800);
}

```

See the *Programmer's Reference* in Appendix A for more information on writing the RS-232 and RS-485 interface methods and writing timer methods.

### *Writing the FPM Driver Work() Routine*

The `Work()` routine is executed when one or more data points declared in the FPM driver are updated. In the `Work()` routine, you write one or more IF-THEN(-ELSE) statements that use the `Changed()` method to evaluate whether the data points in the FPM driver have been updated.

If a data point has been updated, you initialize communication between your FPM and the devices connected to the RS-232 and RS-485 serial interfaces. You can initialize communication with the RS-232 serial interface using the `rs232_ioctl()` method. You can initialize communication with the RS-485 serial port using the `rs485_setparams()` and `rs485_ioctl()` methods.

Once you initialize communication, you can directly write to the RS-232 and RS-485 serial interfaces without using any additional methods. You can also read data point properties using the methods described in *Writing the FPM Application Work() Routine*.

```

void CUFPT_Display::Work()
{
    if (Changed (Line1))
    {
        printf("update for Line1 (%s)\n", Line1->ascii);

        //write Line 1 text to RS-232 interface
        rs232_write(RS232_fd, (Byte *)Line1->ascii,
                  strlen((char*)Line1->ascii));
    }
}

```

### *Writing the FPM Driver OnTimer() Routine*

The `OnTimer()` routine is executed when a timer started with the `Start()` method in the `Initialize()` routine expires. You can use this routine to initialize communication between your

FPM and the devices connected to the RS-232 and RS-485 serial interfaces, read data from the RS-232 or RS-485 interface, write updated values to data points, and read data point properties.

If you started more than one timer using the `Start()` method of the `CFPM_Timer` class, you must first identify the timer that expired using the `m_oTimer.Expired()` method. In addition, you must create custom `OnMyTimer()` routines for each user-defined timers you started with the `START_TIMER()` macro in the `Initialize()` routine.

You can initialize communication with the RS-232 and RS-485 serial interfaces using the methods described in the previous section, *Writing the FPM Driver Work() Routine*. You can write data point values and read data point properties using the methods described in *Writing the FPM Application Work() Routine*. See `rs232_read()` in Appendix A for more information on using the `ReadBytes()` function to read data from the RS-232 interface.

The following code demonstrates how to create an `OnTimer()` routine for an FPM driver.

```
//define buffer size in header file
#define MAX_RXBUFLLEN 512
Byte rxBuf [MAX_RXBUFLLEN];
...
void CRs232Driver::OnTimer()
{
    if (m_oDisplay_InputTimer.Expired())
    {
        memset(rxBuf, 0, MAX_RXBUFLLEN);
        int nBytesRead = ReadBytes(RS232_fd, rxBuf, MAX_RX_BYTE_SIZE);

        //check whether something has been read
        if (nBytesRead >= 1)
        {
            printf ("Read %c from RS232\n", Linel);
            //if something has been read, write it to display device
            rs232_write(RS232_fd, (Byte *)rxBuf, nBytesRead);
        }
    }
}
```

### *Writing the FPM Driver Shutdown() Routine*

The `Shutdown()` routine is executed when the FPM driver is stopped or disabled. In the `Shutdown()` routine, you close the RS-232 or RS-485 interface, stop timers, and perform any required cleanup.

To delete a timer, you can use the `StopTimer()` or `StopAllTimers()` methods. To close the RS-232 interface, you use the `rs232_close()` method. To close the RS-485 interface, you use the `rs485_close()` method.

The following code demonstrates how you can close an RS-232 interface and stop timers in the `Shutdown()` routine:

```
void CUFPT_Display::Shutdown()
{
    m_oDisplay_InputTimer.Stop();
    rs232_close(RS232_fd);
}
```

See the *Programmer's Reference* in Appendix A for more information on stopping timers and closing the RS-232 and RS-485 interfaces.

---

## Compiling an FPM

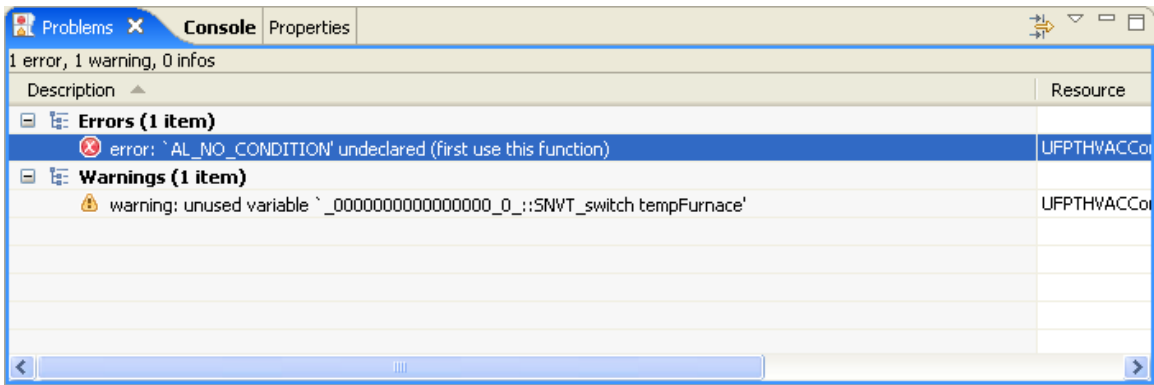
Once you finish writing an FPM, you can compile it with the *i.LON SmartServer 2.0 Programming Tool*. If your code has any errors, they will be listed with any warnings in the **Problems** view at the bottom of the document window. You can click on the errors and warnings listed in this view to debug your FPM. Following the coding guidelines described in this section will help you debug your code.

To compile your FPM, click **File** and then click **Save**. You will upload this file to the SmartServer as described in the next chapter, *Deploying FPMs on a SmartServer*. If the build is not performed, click **Project** and then click **Build Project**. You can then click **Project** and select **Build Automatically** so that your FPM applications are built automatically when you save them.

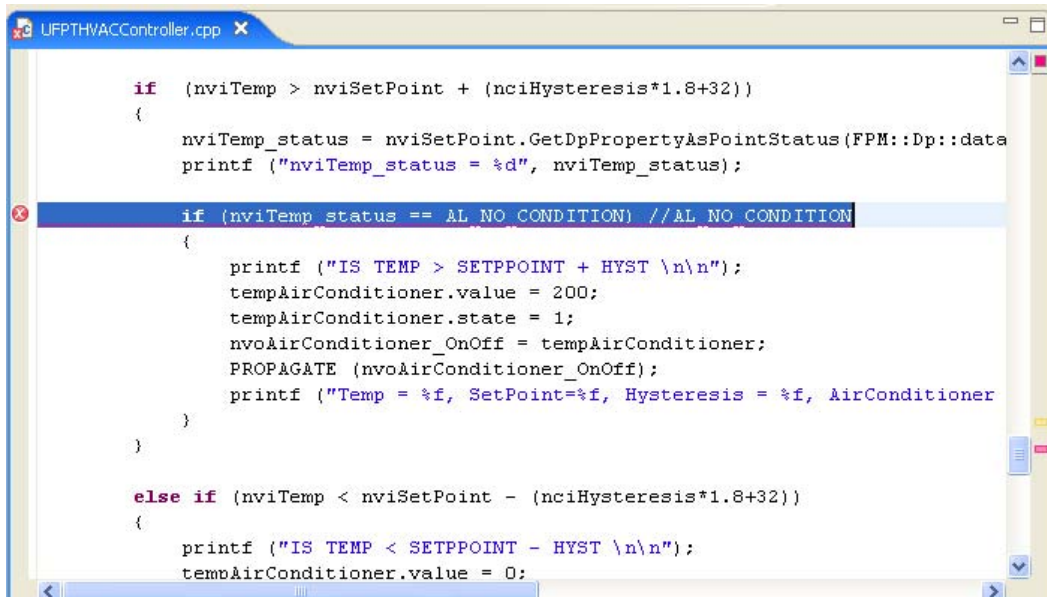
**Note:** If a dialog appears prompting you to enter a license, you need to install the full version of the *i.LON SmartServer 2.0 Programming Tools* on your computer in order to build your FPM application. To order the full version of the *i.LON SmartServer 2.0 Programming Tools*, contact your Echelon sales representative.

### Checking Compile and Warning Errors

If your code has any compile errors or warnings, they will be listed in the **Problems** view at the bottom of the document window. You must resolve the errors and re-compile your FPM to create a successful build and upload it to the SmartServer. You do not need to address the warnings in order to create a successful build, but you should fix them because they typically indicate future bugs.



To resolve an error or warning, click the error or warning in the **Problems** view. The focus should switch to the line of code generating the error or warning, which is marked with an error or warning symbol.



```
UFPTHVACController.cpp
if (nviTemp > nviSetPoint + (nciHysteresis*1.8+32))
{
    nviTemp_status = nviSetPoint.GetDpPropertyAsPointStatus(FPM::Dp::data);
    printf ("nviTemp_status = %d", nviTemp_status);
    if (nviTemp_status == AL_NO_CONDITION) //AL_NO_CONDITION
    {
        printf ("IS TEMP > SETPOINT + HYST \n\n");
        tempAirConditioner.value = 200;
        tempAirConditioner.state = 1;
        nvoAirConditioner_OnOff = tempAirConditioner;
        PROPAGATE (nvoAirConditioner_OnOff);
        printf ("Temp = %f, SetPoint=%f, Hysteresis = %f, AirConditioner
    }
}
else if (nviTemp < nviSetPoint - (nciHysteresis*1.8+32))
{
    printf ("IS TEMP < SETPOINT - HYST \n\n");
    tempAirConditioner.value = 0;
```

Once you correct all the compile errors and the warnings, click **File** and then **Save** to re-compile your code.

**Tip:** You can also check the **Console** view (located to the right of the **Problems** view) to see if there is more detailed information available for a given compiler error.

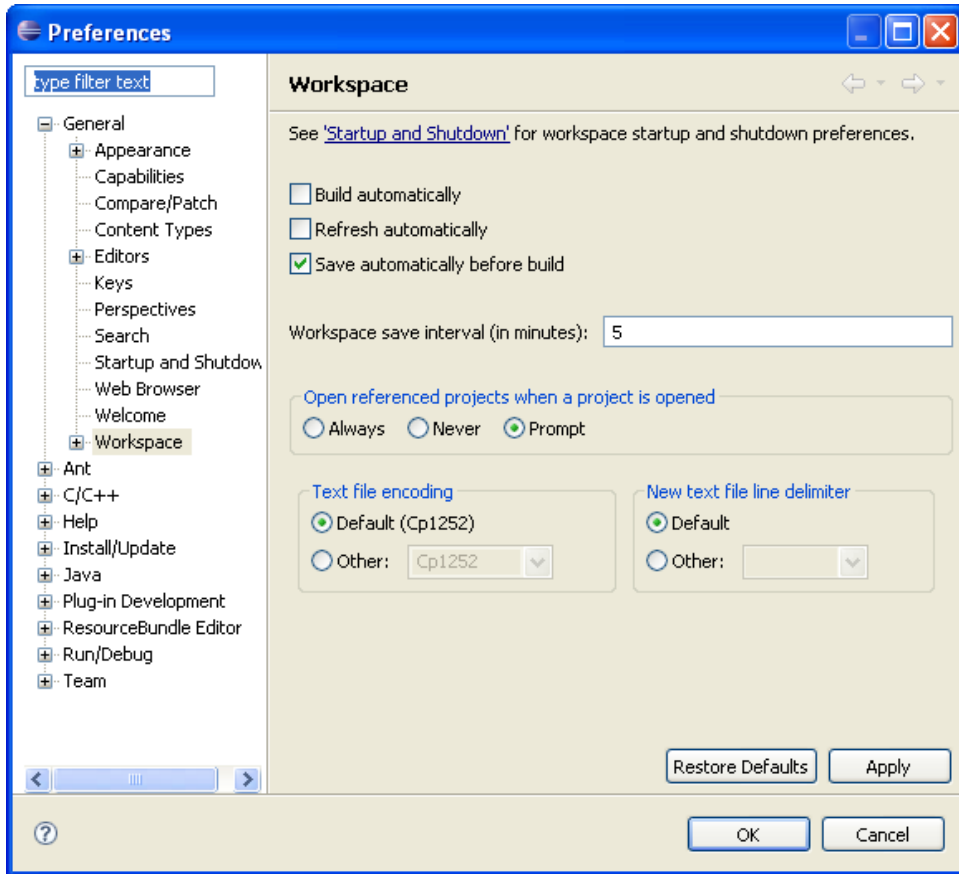
### *Using Non-Latin Characters*

The Eclipse environment uses Cp1252 text encoding by default. If you insert non-Latin characters in your code, you need to implement additional steps to save your FPM project and to display the characters.

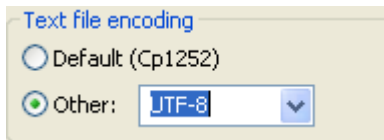
If you want to save an FPM project that has non-Latin characters, but do not need to view the characters in the code, you can change the default text encoding for your FPM project to UTF-8. To do this, follow these steps:

1. Click **Window** and then click **Preferences**. The **Preferences** dialog opens.





2. In the **Text File Encoding** box, click **Other** and then select **UTF-8** from the bottom of the list.



3. Click **OK**. You can now save your FPM project.

If you want to save an FPM project that has non-Latin characters and display the characters in your code, you can change your Windows Regional Settings to the native language of the characters. To do this, close the *i.LON SmartServer 2.0 Programming Tool*, open the Control Panel, click **Regional and Language Options**, select the desired language in the **Standards and Formats** box, and then click **OK**. When you re-open the *i.LON SmartServer 2.0 Programming Tool*, you will observe that the change has been implemented.

---

## Debugging FPMs

The SmartServer uses a VxWorks® real-time operating system to run its embedded applications. If you need a source level debugger (VxWorks 6.2 - Wind River Workbench 2.4) or access to VxWorks system calls not encapsulated in the Echelon FPM API, contact Wind River® sales at [www.windriver.com/company/contact/index.html](http://www.windriver.com/company/contact/index.html) for more information on ordering “WindRiver Platform for Industrial Services V3.2 for MIPS32 Processors”.

If you plan on debugging your FPMs with Wind River Workbench, you need to backup and then delete the current **iLonSystem** image on your SmartServer flash disk, copy the **iLonSystemWdb** or **iLonSystemWbdEnd** image in the LonWorks/iLON/Development/Debug/ES\_Debug.<software version> folder on your computer to your SmartServer flash disk, re-name the **iLonSystemWdb** or **iLonSystemWbdEnd** image on your SmartServer to **iLonSystem**, reboot the SmartServer, create a

debug configuration of your FPM in the *i.LON SmartServer 2.0 Programming Tool* and upload it to your SmartServer, and then connect the Workbench debugger to the **iLonSystemWdb** or **iLonSystemWdbEnd** image on your computer via the target server.

If you are not using Wind River Workbench to debug your FPMs, you can still perform some debugging by adhering to a number of guidelines. These guidelines include connecting the computer running the *i.LON SmartServer 2.0 Programming Tool* to the *i.LON* console port, bracketing your code, and liberally inserting `printf()` statements in your code.

### *Using Wind River Workbench*


Echelon provides **iLonSystemWdb** and **iLonSystemWdbEnd** images for Task Mode and System Mode debugging, respectively.

**iLonSystemWdb** is a bootable VxWorks system image (kernel) with the WDB debugger network connection set to `WDB_COMM_NETWORK` that you can use for Task Mode debugging. Because the `WDB_COMM_NETWORK` connection uses the full VxWorks network stack, using the `WDB_COMM_NETWORK` connection and Ethernet-connected Task Mode debugging is fast and reliable—even over the public Internet. Note that you cannot use the `WDB_COMM_NETWORK` for System Mode debugging. If you need to do System Mode debugging, you must use the **iLonSystemWdbEnd** image.

**iLonSystemWdbEnd** is a bootable VxWorks system image (kernel) with the WDB debugger network connection set to `WDB_COMM_END` that you can use for Task Mode debugging and for System Mode debugging of FPM drivers. System Mode debugging is commonly used for debugging the VxWorks system image, interrupt service routines, and other debugging with interrupts disabled. For more information on Task Mode and System Mode debugging, see the Wind River Workbench documentation.

For a WDB connection to work, you must ensure that UDP port 17185 is open in both directions on all hardware and software firewalls between the debugging host computer and the target SmartServer. This includes firewalls on your host computer and your corporate network.

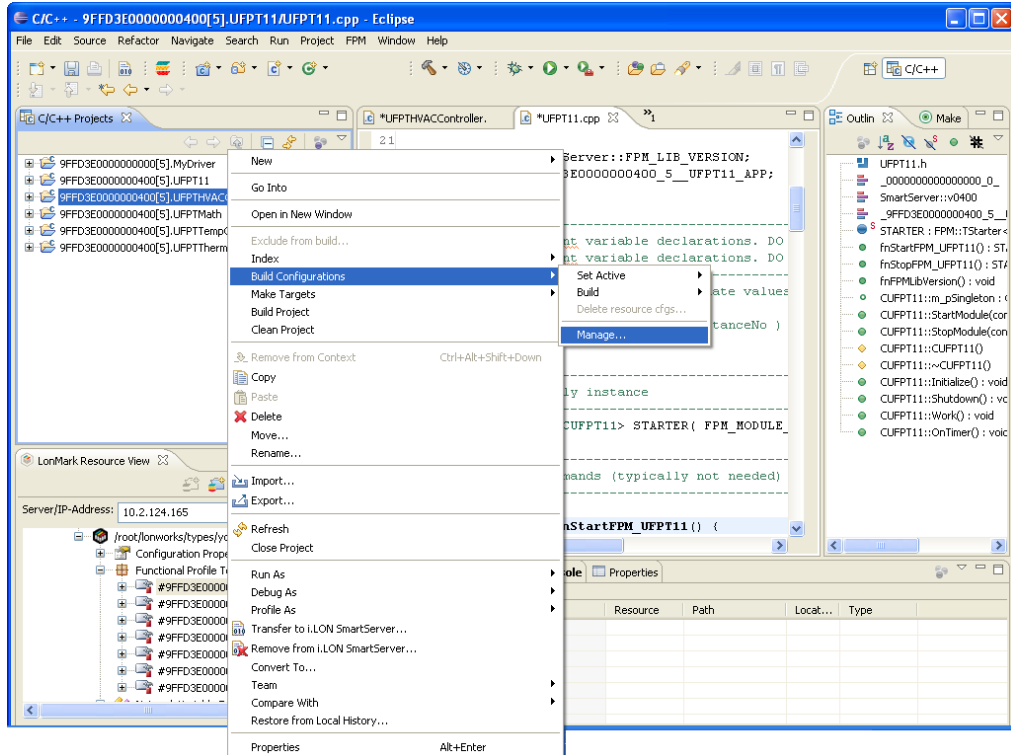
If you are debugging your FPMs with Wind River Workbench, you need to perform the following steps to create a debug configuration of your FPM and connect the *i.LON* system image to Wind River Workbench.

1. Backup the current **iLonSystem** image on the root directory of your SmartServer flash disk. You can copy the **iLonSystem** image to the local drive of your computer, a USB drive, a floppy disk, another removable media, or a shared network drive with read/write permissions. After you create the backup, delete the **iLonSystem** image from the SmartServer flash disk.
2. Copy the **iLonSystemWdb** or **iLonSystemWdbEnd** image from the `LonWorks/iLON/Development/Debug/ES_Debug.<software version>` folder on your computer to the root directory of your SmartServer flash disk.
3. Re-name the **iLonSystemWdb** or **iLonSystemWdbEnd** image you copied to the SmartServer flash disk to **iLonSystem**.
4. Reboot the SmartServer using the *i.LON SmartServer 2.0 Programming Tool*, the SmartServer Web pages, or the SmartServer console application.
  - To reboot your SmartServer using the *i.LON SmartServer 2.0 Programming Tool*, click **FPM**, and then click **Reboot i.LON SmartServer 2.0** (alternatively, you can click the Echelon logo in the menu bar [  ]). The Reboot *i.LON SmartServer 2.0* dialog opens. Enter the IP address or hostname of your SmartServer and then click OK.
  - To reboot your SmartServer using the SmartServer Web pages, right-click the local SmartServer, point to **Setup**, and then click **Reboot** on the shortcut menu. The **Setup – Reboot** dialog opens. Click **Reboot** to start the reboot.

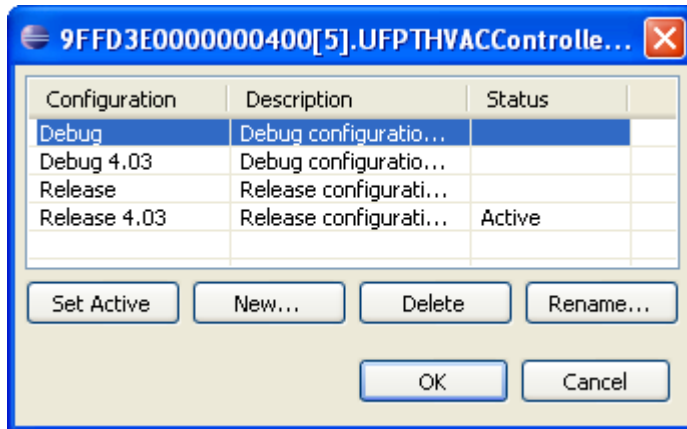
- To reboot your SmartServer using the SmartServer console application, enter the `reboot` command. For more information on using the SmartServer console application, see the *i.LON SmartServer 2.0 User's Guide*.
- Create a debug configuration for your FPM project. You can skip to step 6 if you already have a debug configuration for your FPM. You have already created a debug configuration if you imported and upgraded an existing debug configuration of an FPM project to the **Debug 4.03** format as described in *Upgrading the i.LON SmartServer 2.0 Programming Tool* in Chapter 2, or if you selected the **Debug 4.03** or **Debug** configurations when you created the FPM project as described in the *Creating New FPM Projects* section earlier in this chapter.

To check whether there is a debug configuration created for your FPM, and then create a debug configuration, if necessary, follow these steps:

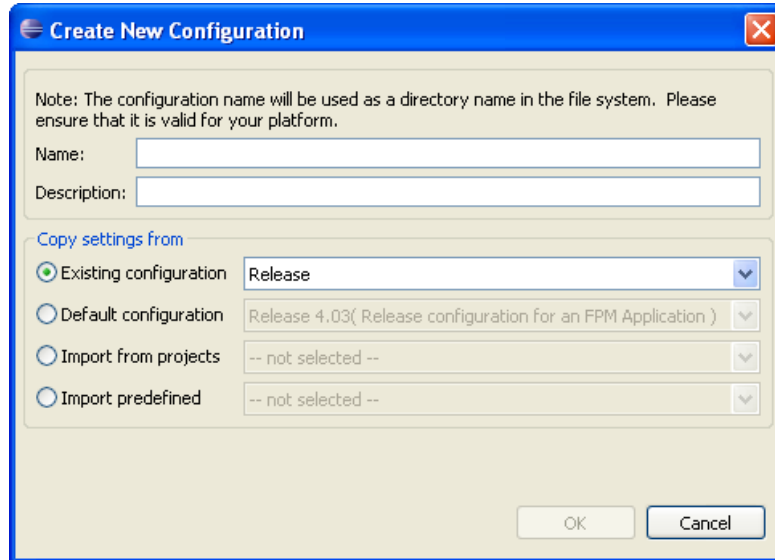
- Right-click the FPM project, point to **Build Configurations**, and click **Manage**.



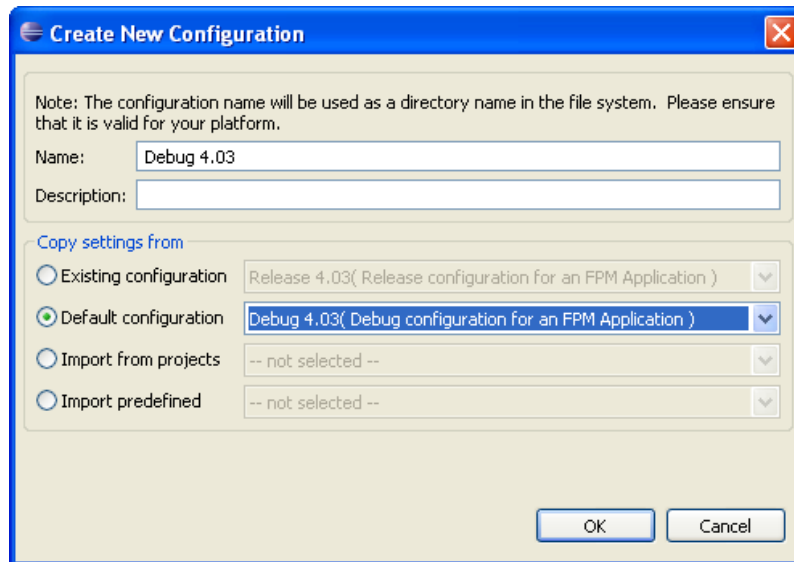
- The **Manage Configurations** dialog opens.



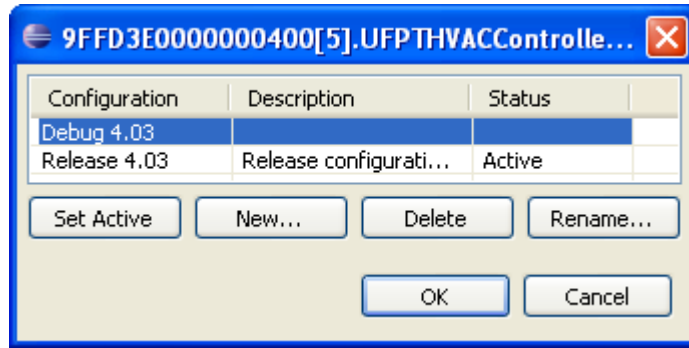
- c. This dialog lists the build configurations available for your FPM. If the **Debug 4.03** configuration is listed (or **Debug** if you plan on running your debug configuration on a SmartServer 1.0), you can skip to step 6; otherwise continue to step d.
- Note:** If you are debugging an FPM to be run on a SmartServer 1.0, select or enter **Debug** in the following steps instead of **Debug 4.03**.
- d. Click **New** in the **Manage Configurations** dialog. The **Create New Configurations** dialog opens.



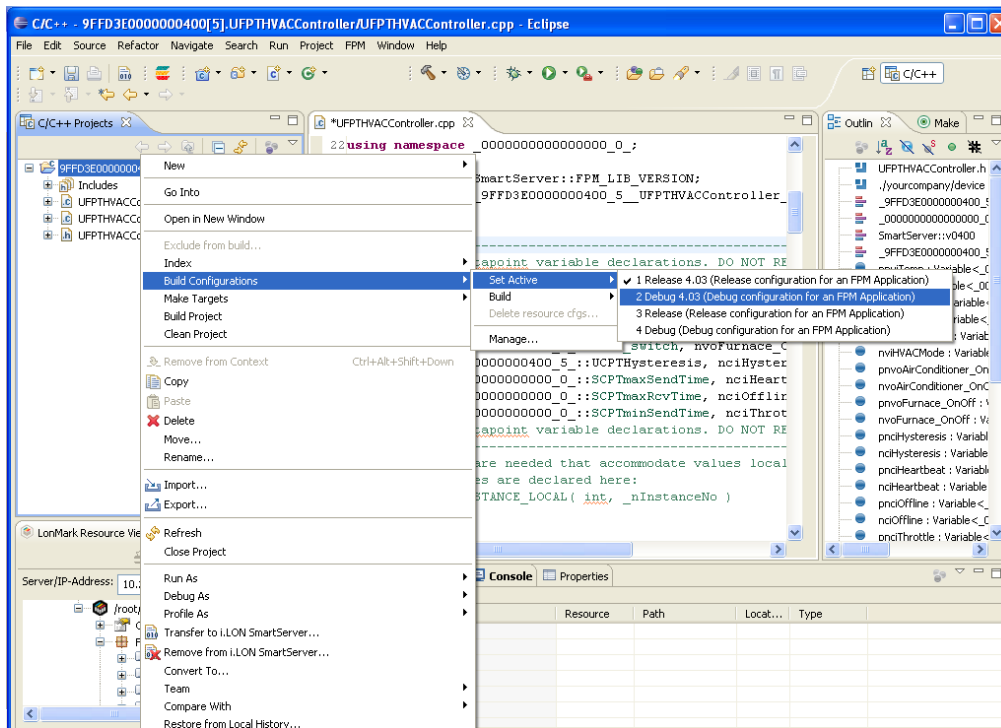
- e. In the **Name** property, enter **Debug 4.03**.
- f. Select the **Default Configuration** option, and then select **Debug 4.03**.



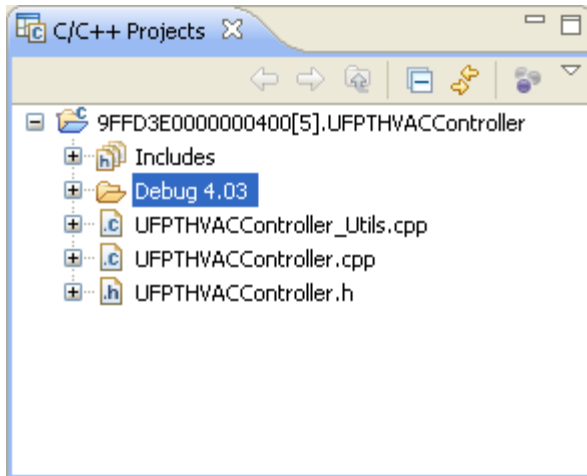
- g. Click **OK** to return to the **Manage Configurations** dialog.



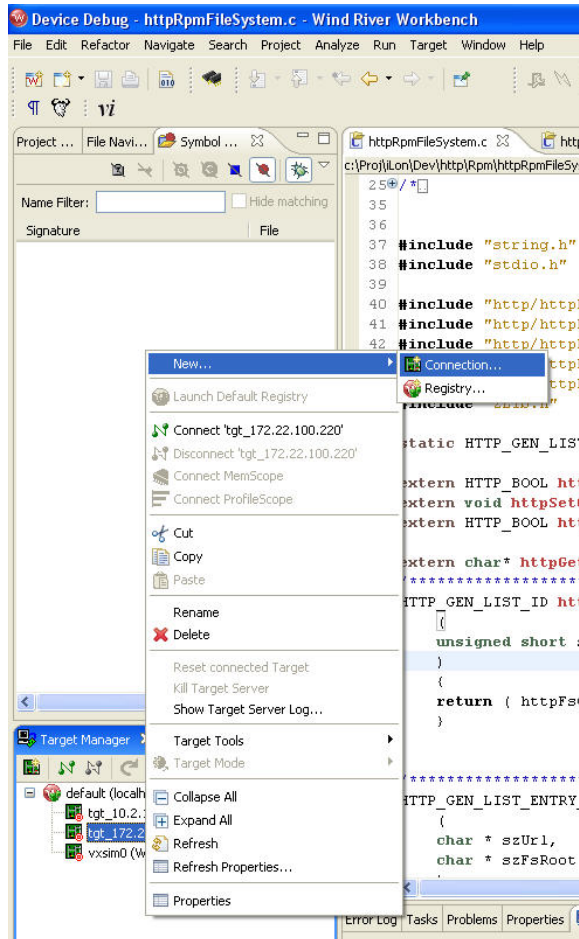
- h. Click **OK** to return to the **C/C++ Projects** view.
- Set the **Debug 4.03** or **Debug** configuration as the active configuration so that it is built automatically when you build your FPM. To do this, right-click the FPM project, point to **Build Configurations**, point to **Set Active**, and then click **Debug 4.03** or **Debug**.



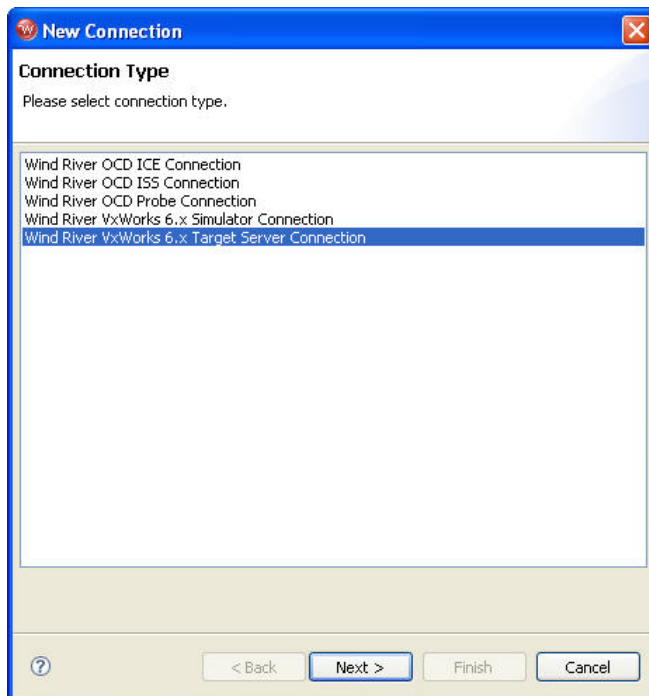
- Build your FPM project, as described in *Compiling an FPM* earlier in this chapter. A **Debug 4.03** or **Debug** folder now appears in the **C/C++ Projects View** under the **Includes** folder.



8. Upload the debug configuration of your FPM to your SmartServer. To do this, expand the **Debug 4.03** or **Debug** folder, right-click the `<company program ID>.UFPT<FPM name>.app || .drv` file and then click **Transfer to i.LON SmartServer** in the shortcut menu. In the **Install FPM Module** dialog, enter the IP address or hostname of your SmartServer in the **Host** box, select the **Reboot** check box, click **Finish**, and then click **Yes** to confirm the rebooting of your SmartServer.
9. Deploy the debug configuration of the FPM on your SmartServer as described in *Adding FPM Devices to the SmartServer* in Chapter 6.
10. Connect the Workbench debugger to the **iLonSystemWdb** or **iLonSystemWdbEnd** image on your computer via the target server, following these steps:
  - a. Verify that the WorkBench registry is running.
  - b. Right-click anywhere in the **Target Manager** view, point to **New**, and then click **Connection...** on the shortcut menu.

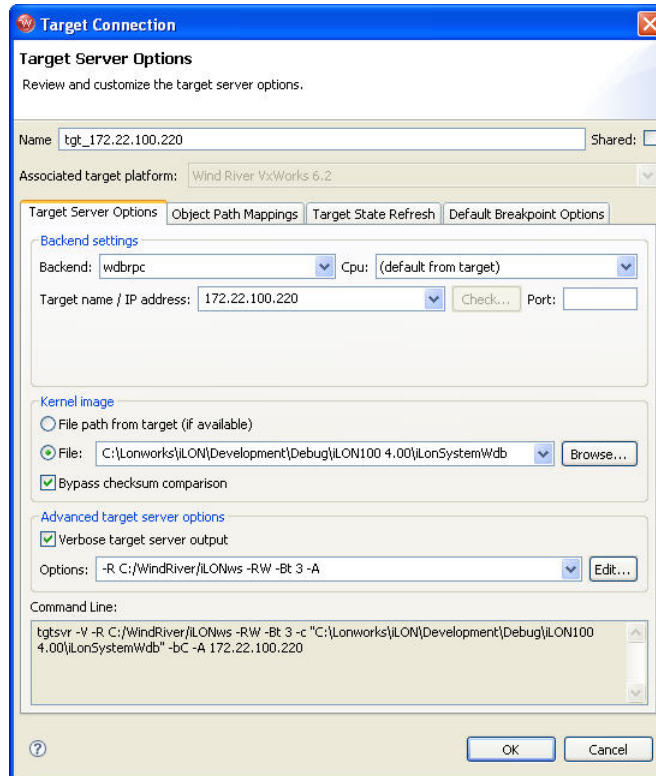


- c. The **New Connection** dialog opens. In the **Connection Type** window, select the **WindRiver VxWorks 6.x Target Server Connection** and then click **Next**.

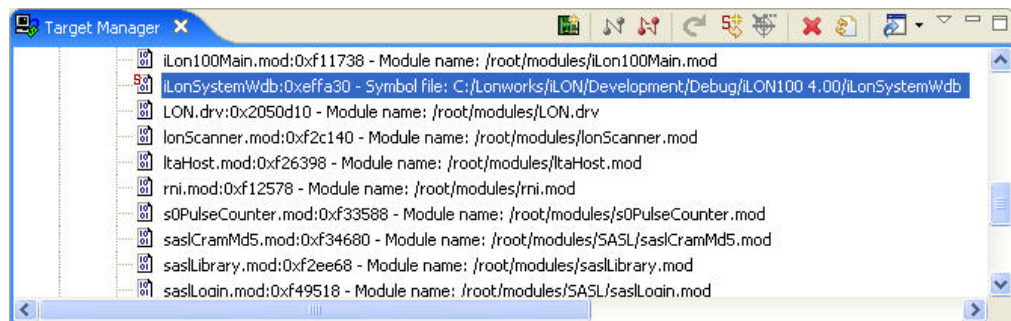




- d. In the **Target Server Options** window, enter the IP address or hostname of the SmartServer on which the FPM to be debugged is installed.
- e. In the **Kernel Image** box, click **File**, click **Browse**, and then browse to the LonWorks/iLON/Development/Debug/ES\_Debug.<software version> folder on your computer.



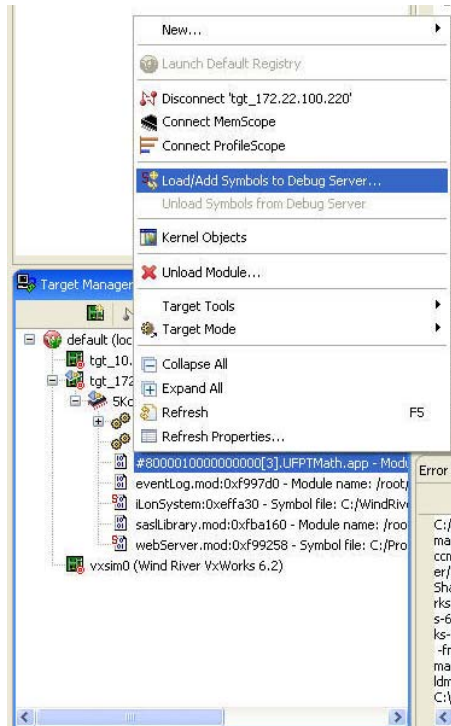
- f. Click **Finish**. The **iLonSystem** image and the FPM executable module (.app or .drv extension) appear in the **Target Manager** view.



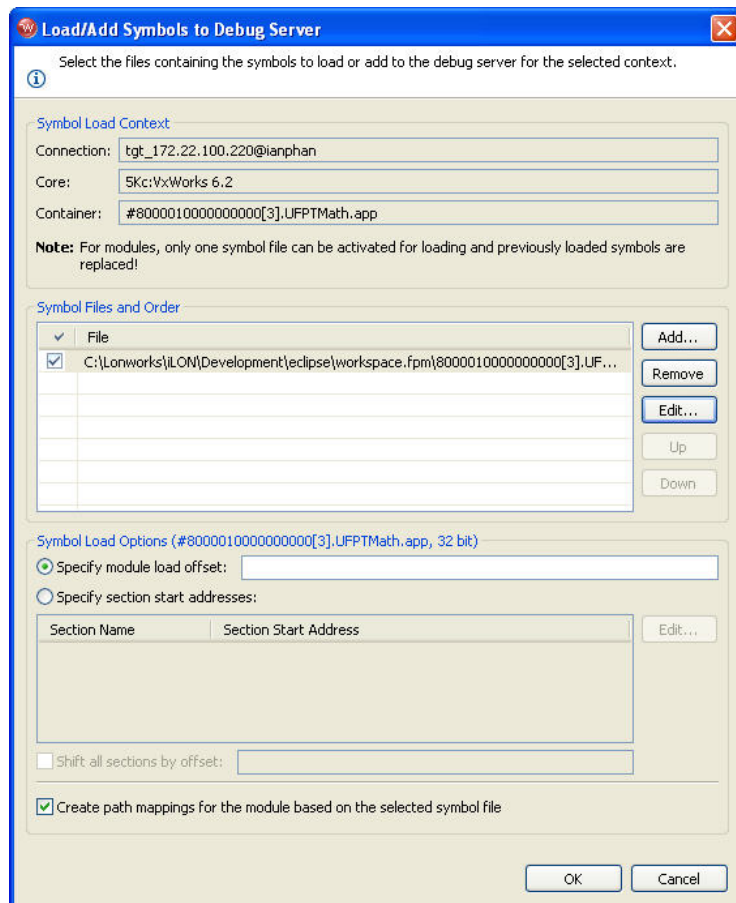
11. Add the symbols in your FPM to the target server, following these steps:

- a. In the **Target Manager** view, right-click your FPM executable module (.app or .drv extension), and then click **Load/Add Symbols to Debug Server** on the shortcut menu.

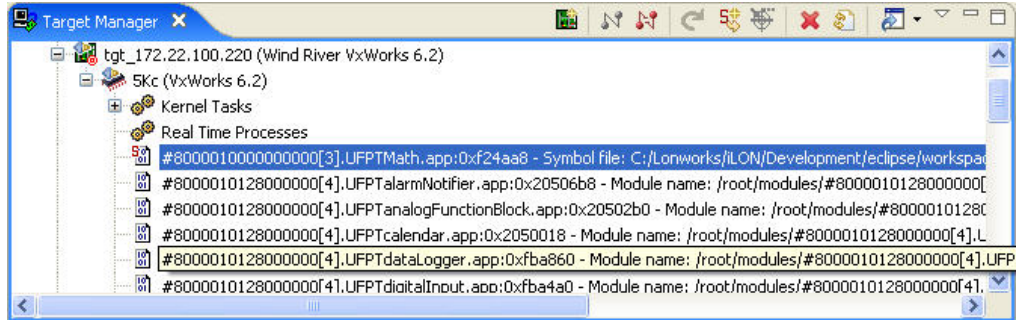




- b. The **Load/Add Symbols to Debug Server** dialog opens. In the **Symbol Files and Order** box, click **Add** and then browse to the debug configuration of your FPM.

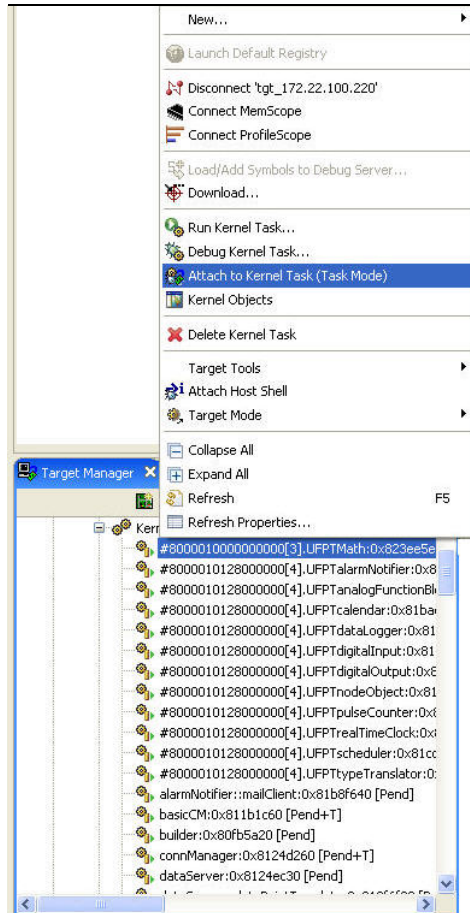


- c. Click **OK**. The symbols in your FPM are now loaded in the **Target Manager** view. Symbol icons should appear on the icons representing the FPM executable module and the **iLonSystem** image.

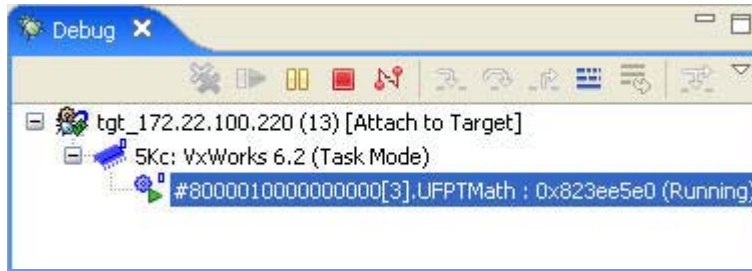


12. Use WindRiver Work Bench to debug your FPMs, following these steps:

- a. Expand the **Kernel Tasks** icon.
- b. In the **Target Manager** view, right-click the FPM task and click **Attach to Kernel Task (Task Mode)** on the shortcut menu.



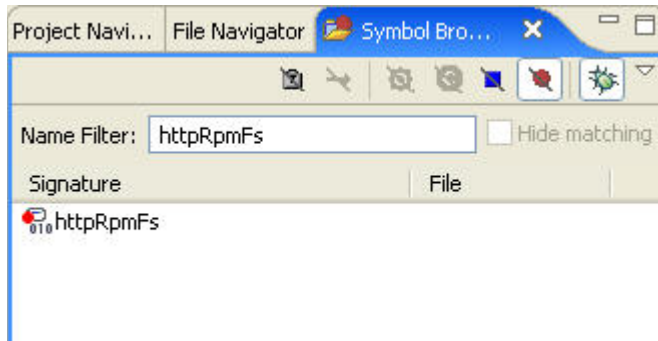
- c. The **Debug** view opens, and you can now use the **Symbol Browser** view search for symbols in your FPM code.



- d. If you are using the **iLonSystemWdbEnd** image and want to switch to **System Mode**, right-click the FPM task, point to **Target Mode**, and click **System Mode** on the shortcut menu.



- e. In the **Symbol Browser** view, first verify that the Debug icon (🔍) is enabled. In the **Name Filter** box, you can search for the symbol for which you want to set a breakpoint.



- f. Once the symbol appears in the **Symbol Browser** view, double-click it. The source file view displays the routine to be debugged.
- g. In the source file view, double-click the line number in the routine to set a breakpoint.

### Using FPM Development Guidelines

You can perform some debugging of your FPMs by adhering to the following guidelines when developing your FPM:

- Physically connect the computer running the *i.LON* SmartServer 2.0 Programming Tool to the *i.LON* console port using an RS-232 null modem cable. This enables you to use a Terminal emulator such as Windows HyperTerminal to view the *i.LON* console port and debug your FPMs during runtime. After the FPM is initialized you can use Telnet to view the *i.LON* console port.
- Back up the FPM project frequently. Always make a back up after you make significant changes to an FPM application and successfully compile it.
- Bracket comments around those portions of the FPM application that you have written. For example, you can do the following:

```
// mycode - begin -----
out1 = in1 + in2;
// mycode - end -----
```

- Add your user help functions to the UFPT<FPM>\_Utils.cpp file (this file is created when you create a new FPM project). This further isolates your code for debugging, and it enables you to port the code over to another FPM project.
- Insert `printf()` statements in your code frequently. This enables you to do some debugging with the console port of the *iLON* during runtime, as the console port will receive the `printf()` statements. The following example demonstrates a `printf()` statement that you can use to debug your code.

```
printf("[%s %i] value of %s: %d",
      __FILE__,
      __LINE__,
      in1.GetDpPropertyAsString(FPM::Dp::cfgUCPTname),
      *in1);
```

Note that console port displays the status of your FPMs during a reboot.

It is especially important to follow these guidelines because the compiler errors you may receive may only have a generic description that does not indicate which line of code caused the error. In addition, the errors may not appear on the actual line of code causing the error; instead, an error may appear one or two lines above the incorrect code.

## Using SNMP Support

You can create an FPM that provides SmartServer system information via the Simple Network Management Protocol (SNMP). To do this, you use the *iLON* SmartServer 2.0 Programming Tool and a VxWorks 6.2 Development Tool that includes the SNMP v1/v2c or v3 agent components (for example, PID 3.2 including the GNU compiler). The SmartServer does not support eXtensible agent (AgentX) components.

The following table lists the default SmartServer system information that you can provide over SNMP:

<i>System</i>	Description and name of the Smart Server and other information.
<i>Interfaces</i>	IP interface information such as speed and address.
<i>IP address table</i>	Internal IP address table on the SmartServer.
<i>Route Entries</i>	The routing table of the SmartServer.

You can use the VxWorks SNMP API provided by a VxWorks 6.2 Development Tool to change the management information base (MIB-II) configuration that is compiled in the SmartServer's **iLonSystem** image. You can add or delete MIB-II tree elements, and then map these elements to data points in an FPM application running on the SmartServer. You can then read and write to these data points over SNMP. You can also the SNMP trap service to send messages or alarms from an FPM application running on a SmartServer to an external SNMP trap server.

For more information on SNMP programming, see the *Wind River SNMP for VxWorks Programmer's Guide 10.0*.

The **LonWorks\iLON\Development\eclipse\workspace.fpm** directory on your computer contains example SNMP client and server FPMs. You can upload these FPMs to your SmartServer and run them, or you can edit the project files for each example to see how they are implemented.

## Example FPM Applications and Drivers

The **LonWorks\iLON\Development\eclipse\workspace.fpm** directory on your computer contains one FPM application example and one FPM driver example, in addition to the example SNMP client and server FPMs mentioned in the previous section. You can upload these FPMs to your SmartServer and run them, or you can edit the project files for each example to see how they are implemented. The FPM examples are listed below:

- **Rs232Driver.** This FPM driver demonstrates how to read and write data from a RS-232 interface.

- **Math.** This FPM application adds two **SNVT\_count** input data points when one of the data point values changes, and it writes the sum in a **SNVT\_count** output data point.



## Deploying FPMs on a SmartServer

This chapter describes how to use the *i.LON SmartServer 2.0 Programming Tool* to upload FPMs to one or more SmartServers. It explains how to select a network management service (LNS or Standalone) for running your LONWORKS network. It describes how to create, commission, and connect, and test FPM devices on the SmartServer. It describes how to create a custom configuration Web page for FPM applications. It explains how to update FPM applications. It describes how to deploy FPMs on multiple SmartServers and it describes how to deploy licensed FPMs.

---

## FPM Deployment Overview

After you write and compile an FPM application or driver, you can deploy it on your SmartServer. This enables you to apply the algorithms defined in the FPM application or FPM driver to the data points on the SmartServer. Deploying an FPM application entails uploading the FPM to your SmartServer, adding a device representing the FPM application on the SmartServer, commissioning the FPM device if you are going to bind the data points in the FPM application with LonWorks connections, testing the FPM application, and then connecting the data point in the FPM device with LONWORKS connections or Web connections. Deploying an FPM driver entails just uploading the FPM to your SmartServer.

You can upload FPMs to one or more SmartServers that have an FPM programming license installed on them using the *i.LON SmartServer 2.0 Programming Tool*. After an FPM application has been uploaded to a SmartServer, you need to verify that you have selected a network management service mode (LNS or Standalone) for running your LONWORKS network.

Once you have selected a network management service, you can create an FPM device on the SmartServer. To do this, you add a new internal device to the **LON** channel in the SmartServer tree. If you are integrating your FPM application with another LNS application such as the LonMaker tool, the internal FPM device must use a static interface. To use a static interface for your FPM, you select the device interface (XIF) file from the root/lonworks/import/<YourCompany> folder on the SmartServer flash disk. This is the XIF file that you generated for your FPM with the *i.LON LonWorks Interface Developer tool* (see Chapter 4 for more information on creating XIF files for FPMs).

If you are running your network with the SmartServer operating as a standalone network manager, the internal device can use a static or dynamic interface. To use a dynamic interface, you select the SmartServer's v40 XIF from the root/lonworks/import/Echelon/iLON100 folder, and you then add a dynamic functional block to the device that uses a UFPT representing your FPM application.

If you are running your LONWORKS network in LNS mode (**LNS Auto** or **LNS Manual**) and you plan on using LONWORKS connections to bind the data points in your FPM application with the data points on the internal SmartServer device, on another FPM device, or on the external devices connected to the SmartServer, you must first commission your FPM device. You can commission your FPM device with the SmartServer or with an LNS application such as the LonMaker tool.

You can test that the FPM application is functioning properly by adding the data points declared in the FPM device to the **View – Data Points** Web page. You can then change the values of the input data points and observe whether the output data points are updated accordingly. Note that if FPM programming is not licensed on the SmartServer, the SmartServer will not process changes made to the FPM data points, and the FPM data points will become unavailable in the **View – Data Points** Web page.

Once you verify that the FPM application is working, you can connect the data points in your FPM application with LONWORKS connections or Web connections. You can then use the **View – Data Points** Web page, a custom FPM configuration Web page, or an LNS application such as the LonMaker tool to test that the connections are updating the FPM data points accordingly.

After you have deployed FPM applications on a development SmartServer, you can deploy the FPM applications you have developed on multiple SmartServers.

### Notes:

- The full version of the *i.LON SmartServer 2.0 Programming Tools* must be installed on your computer to upload an FPM to a SmartServer. The full version of the *i.LON SmartServer 2.0 Programming Tools* is included on the *i.LON SmartServer 2.0 Programming Tools DVD*. To order the *i.LON SmartServer 2.0 Programming Tools DVD* (Echelon part number 72111-409), contact your Echelon sales representative.



- An FPM programming license must be installed on your SmartServer in order for an FPM to function on the SmartServer. If FPM Programming is not licensed on a SmartServer, the SmartServer will not process the tasks defined in the FPM application. You can order a FPM programming license from the *i.LON SmartServer 2.0* Web site at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate).
- The device interface (XIF) file that you created for your FPM must be in the root/lonworks/import/<YourCompany> on the SmartServer flash disk. Otherwise, you will not be able to create FPM devices on that SmartServer that uses a static interface. In addition, the XIF file must be in the lonworks/import/<YourCompany> folder on your computer. See Chapter 4, *Creating FPM Device Interface (XIF) Files*, for more information on how to create a XIF for your FPM and copy it from the on your computer to a SmartServer.

---

## Uploading FPM Applications and Drivers

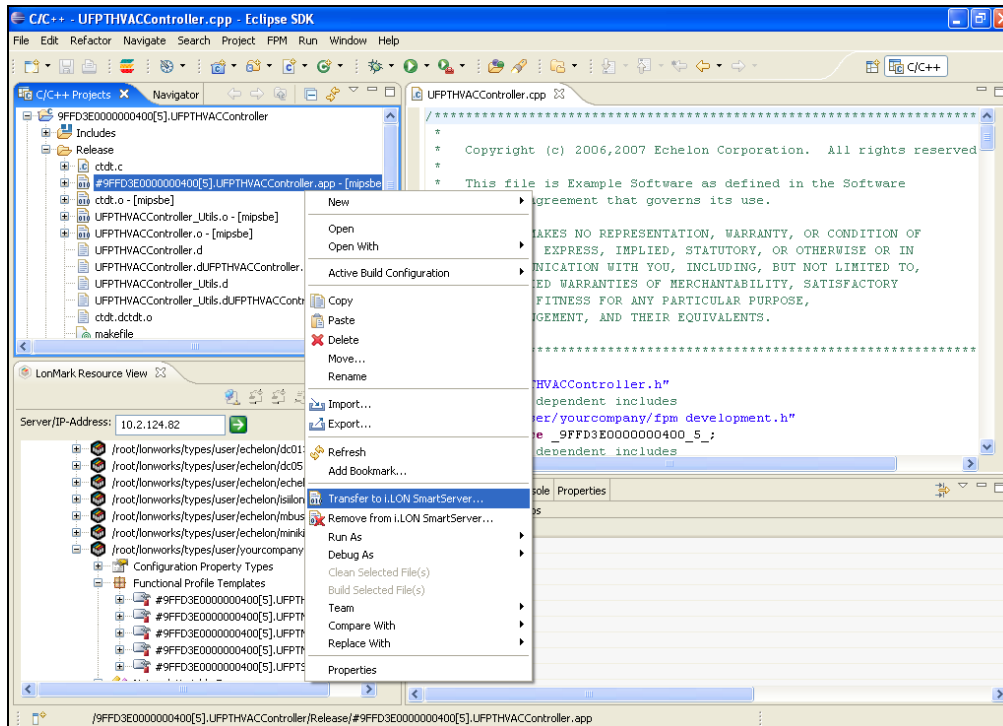
You can upload FPMs to SmartServers that have an FPM programming license installed on them. To do this, you use the *i.LON SmartServer 2.0* Programming Tool or an FTP client such as Internet Explorer 7 to transfer the FPM executable module (**.app** or **.drv** extension) to the root/modules/User/<YourCompany> folder of each SmartServer on which the FPM is to be used.

Typically, if you are deploying your FPM on a development SmartServer you will use the *i.LON SmartServer 2.0* Programming Tool to transfer the FPM module. If you are deploying your FPM on multiple SmartServers in the field, you will use an FTP client because you also need to transfer resource files, device interface (XIF) files (if the FPM uses static functional blocks, and custom FPM configuration Web pages (if created). For more information on uploading the FPM executable module and other required files to multiple SmartServers via FTP, see *Deploying FPMs on Multiple SmartServers* later in this chapter.

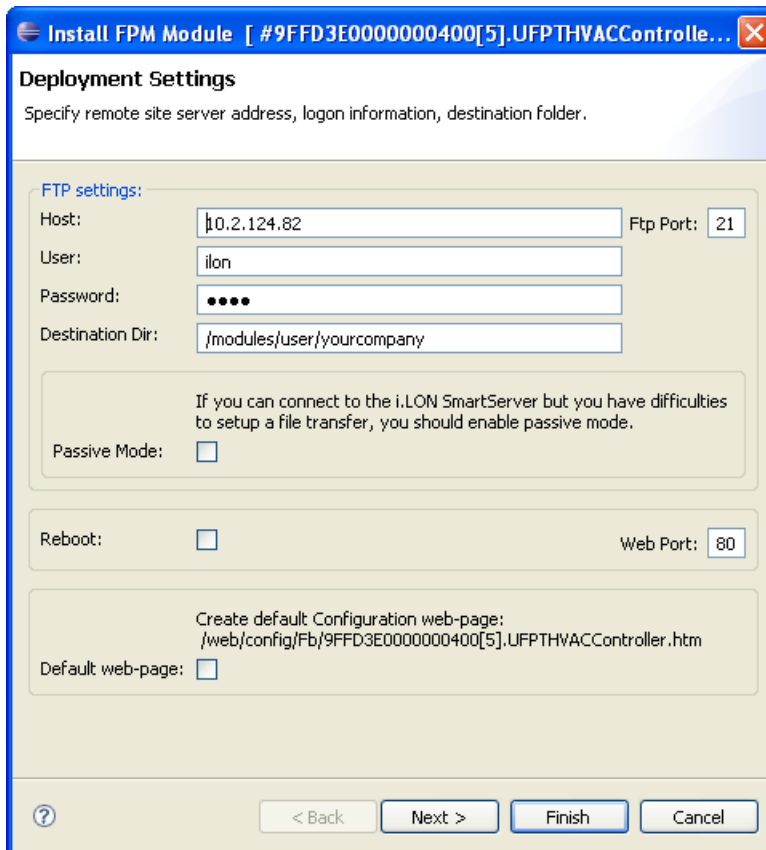
**Note:** As of release 4.01 of the *i.LON SmartServer 2.0* Programming Tool, you do not need to reboot your SmartServer to initialize an FPM application. Once a new or updated FPM application has been uploaded to the SmartServer, it is automatically initialized, and it will execute its algorithms upon data point updates. You must still reboot the SmartServer to initialize an FPM driver.

To upload an FPM application or driver to a SmartServer with the *i.LON SmartServer 2.0* Programming Tool, follow these steps:

1. Create a User/<YourCompany> folder under the root/modules folder on the SmartServer flash disk if one does not already exist. This is where the executable module generated by the *i.LON SmartServer 2.0* Programming Tool should be stored.
2. Start the *i.LON SmartServer 2.0* Programming Tool if it is not already running. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0* Programming Tool opens.
3. In the **C/C++ Projects** view, expand the **Release** folder, right-click the <company program ID>.UFPT<FPM name>.app || .drv file and then click **Transfer to i.LON SmartServer** in the shortcut menu.



4. The **Install FPM Module** dialog opens with the **Deployment Settings** window.

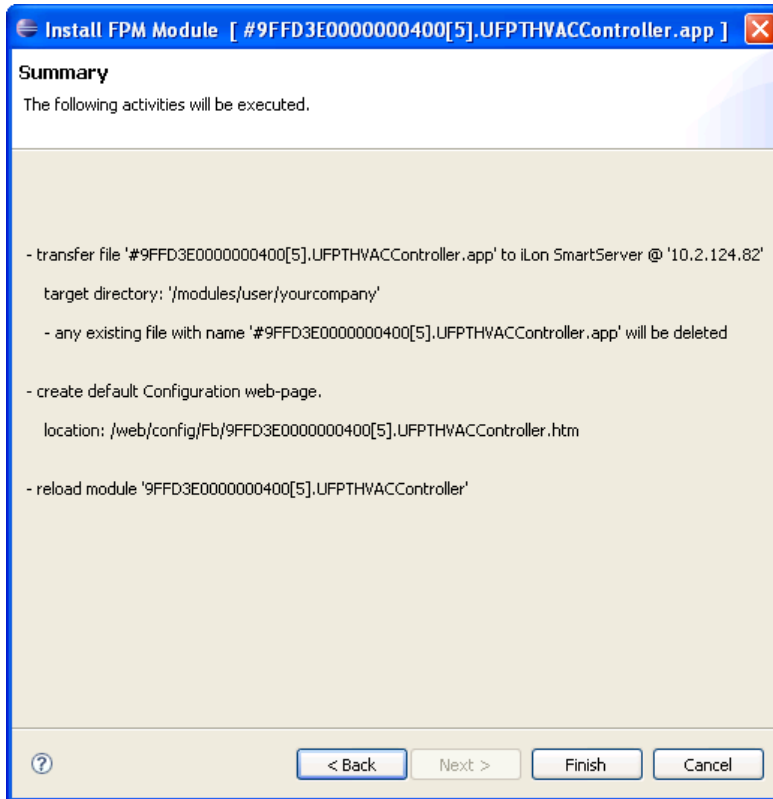


5. Enter the following properties:

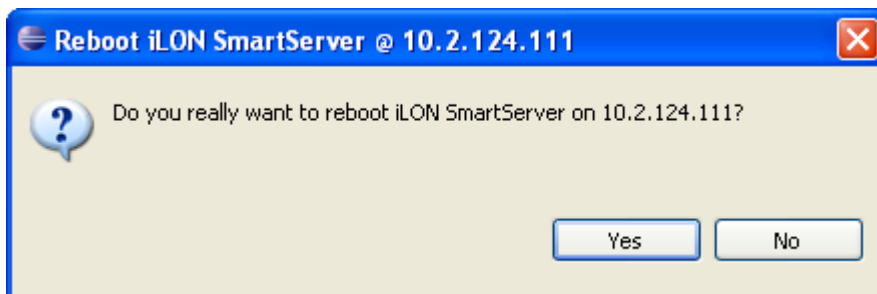
### **FTP Settings**

<i>Host</i>	Enter the IP address or hostname of the SmartServer to which the FPM is to be uploaded. The default is the IP address or hostname entered in the <b>LonMark Resource View</b> .
<i>FTP Port</i>	Enter the port the SmartServer uses for FTP communication. The default FTP port is <b>21</b> .
<i>User</i>	Enter the user name to log in to the SmartServer. The default user name is <b>ilon</b> .
<i>Password</i>	Enter the password to log in to the SmartServer. The default password is <b>ilon</b> .
<i>Destination Directory</i>	Enter the location on the SmartServer flash disk where the FPM application is to be stored. By default, the FPM application is stored in the root/modules folder on the SmartServer flash disk. You should create a User/<YourCompany> directory under the root/modules folder to store your FPMs.
<i>Passive Mode</i>	Select this check box only if an FTP connection cannot be made. This enables your computer to initiate the connection with the SmartServer FTP server. The FTP server will listen and wait for the connection, rather than initiate it, upon receipt of a transfer command. This option is useful if your computer is behind a firewall that blocks the connection initiated by the FTP server (the firewall may see the connection request as an attack) while in active mode. This check box is cleared by default.
<i>Reboot</i>	Select this check box to reboot the SmartServer after the FPM has been uploaded to it. This check box is cleared by default.  If you are deploying an FPM driver, you must reboot your SmartServer to initialize it.  If you are deploying an FPM application, you do not need to reboot your SmartServer to initialize your FPM application. Once a new or updated FPM has been uploaded to the SmartServer, it is automatically initialized, and it will execute its algorithms upon data point updates.
<i>Web Port</i>	The port your SmartServer uses to serve HTTP requests (SOAP and WebDAV). The default value is <b>80</b> , but you may change it to any valid port number. Contact your IS department to ensure your firewall is configured to allow access to the server on this port.
<i>Default Web Page</i>	Creates a default configuration Web page for your FPM application in the root/web/config/FB folder on the SmartServer flash disk. You can then use the <i>i.LON Vision 2.0</i> software to customize this FPM configuration Web page by adding <i>i.LON Vision 2.0</i> objects to it.  Once you publish the FPM configuration Web page, you can click the <b>General</b> button above the navigation pane on the left side of the SmartServer Web interface, click the functional block representing your FPM application, and use the configuration Web page to read and write values to the data points in your FPM application.  See <i>Creating FPM Configuration Web Pages</i> in this chapter for more information.

6. Click **Next** to open the Summary window, or click **Finish** to begin uploading your FPM to your SmartServer and skip to step 7.



7. This window lists the tasks to be performed, which consists of uploading the FPM to the root/modules directory on the SmartServer flash disk, deleting any existing module with the same name of the FPM being uploaded, rebooting the SmartServer if you selected the **Reboot** check box in the Deployment Settings window in step 4, and optionally creating a default configuration Web page if you selected the **Default Web Page** check box. Click **Finish**.
8. If you selected the **Reboot** check box in the Deployment Settings window in step 4, the **Reboot iLON SmartServer 2.0** dialog opens and prompts you to confirm that the SmartServer selected in the Deployment Settings window in step 4 is to be rebooted.



9. Click **Yes**.
10. The FPM executable module (.app or .drv extension) is uploaded to your SmartServer. You can use the console port to verify that the FPM is being uploaded to your SmartServer. If you are updating an existing FPM application on your SmartServer, the current module is stopped and unloaded, and the updated module is then loaded and initialized.

If FPM Programming is not licensed on your SmartServer, the following urgent messages will appear when the SmartServer attempts to load and start the FPM executable module:

```

[STARTING]      "#8000010000000000[3].UFPTHVACController"
*Urgent* FPM license key is invalid: file Echelon1FPMLic.xml
*Urgent* FPM feature is not properly licensed: file Echelon1FPMLic.xml
failed
Error while starting FPM UFPTHVACController: Can't create FPM-task.
*Urgent* [FP]<??-01E35>Fatal error occurred while initializing!
*Urgent* [FP]<??-01E39>FPM-'#8000010000000000[3].UFPTHVACController': Can't crea
te FPM-task.

```

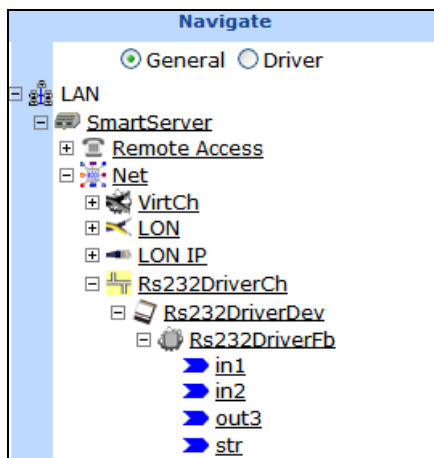
To order an FPM programming license for your SmartServer, go to the *i*.LON SmartServer 2.0 Web site at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate).

### Deploying FPM Applications

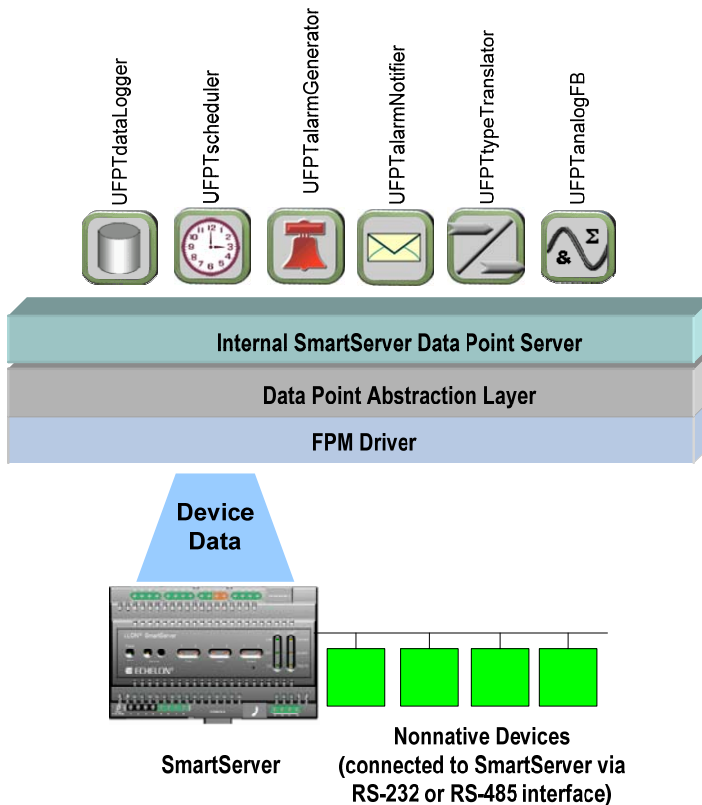
If you are deploying an FPM application, your FPM application is initialized once it has been uploaded to the SmartServer. You can skip the rest of this section and follow the subsequent sections in this chapter to create, commission, connect, and test your FPM application on the SmartServer and create a custom configuration Web page for your FPM application.

### Deploying FPM Drivers

After you upload your FPM driver to the SmartServer and reboot the SmartServer, you can open the SmartServer Web interface and observe that a channel representing your FPM driver has been added to the network tree of the target SmartServer. You can expand the FPM driver channel, expand the FPM driver device (representing the nonnative device connected to the SmartServer's RS-232 or RS-485 interface), and then expand the virtual functional block to show the data points in your FPM driver.



You can now monitor and control the FPM driver data points using the SmartServer's built-in applications and your custom Web pages, and you can connect them to LONWORKS data points using Web connections and Type Translators. In either case, the SmartServer's internal data server sends and receives data point updates to and from the nonnative device via the FPM driver.




---

## Selecting a Network Management Service

Before you create your FPM devices on the SmartServer, you need to verify that you have selected a network management service mode (LNS or Standalone) for running your LONWORKS network.

- In **LNS mode (LNS Auto or LNS Manual)**, the SmartServer transmits network messages to devices through an LNS server, and the SmartServer and the devices connected to it communicate in a peer-to-peer manner. You must use LNS mode if you plan on using LONWORKS connections to bind the data points in your FPM application to other data points. You cannot use LNS mode if your FPM uses a dynamic interface.
- In **Standalone mode**, the SmartServer is the network manager. It directly transmits all network messages to the devices connected to it, and the network functions as a master-slave system, where the SmartServer is the master to the slave devices. You can use standalone mode to operate a small, single-channel network that does not require LNS services, LONWORKS connections, or connections to other network management tools. Networks running in standalone mode are limited to a maximum of 200 devices (for FT-10 networks, you need to attach a physical layer repeater to the network to exceed the 64-device limit posed by the physical channel). FPM devices can use static or dynamic interfaces when the network is running in Standalone mode.

### Using LNS Network Management Services

To configure the SmartServer to use LNS network management services for managing a LONWORKS network, follow these steps:

1. Commission the SmartServer with the LonMaker tool, LNS tree, or another LNS application. For more information on installing the SmartServer, see Chapter 12 of the *i.LON SmartServer 2.0 User's Guide*.

2. Install the Echelon Enterprise Services 2.0 (EES 2.0) and LNS Server Service Pack 5 on an LNS Server (running LNS Turbo Server [version 3.2] or newer) from the *i.LON SmartServer 2.0 DVD* or the *i.LON SmartServer 2.0 Programming Tools DVD*. See Chapter 1 of the *Echelon Enterprise Services 2.0 User's Guide* for how to perform these installations.
3. Add an LNS Server to the LAN and then synchronize the network attached to your SmartServer to an LNS network database. See *Using the LNS Proxy Web Service* in Chapter 3 of the *Echelon Enterprise Services 2.0 User's Guide* for how to do this.

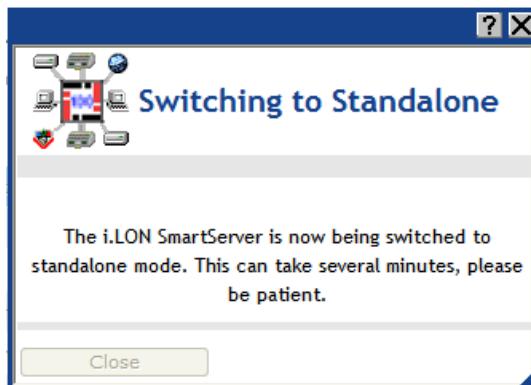
### Using Standalone Network Management

To manage a LONWORKS network using the SmartServer as a standalone network manager, follow these steps.

1. Click the **Driver** button located directly above the navigation pane.
2. Click the network icon in the SmartServer tree. The **Setup - LON Network Driver** Web page opens.
3. In the **Network Management Service** property, select **Standalone**.

Lon Network Property	Value
Icon	iLonNS
Hidden	<input type="checkbox"/>
Network Management Service	<input checked="" type="radio"/> Standalone <input type="radio"/> LNS Auto <input type="radio"/> LNS Manual <input type="checkbox"/> Delete Items Hidden in LonMaker
LNS Server	<input type="text"/>
LNS Network	<input type="text"/> <input type="button" value="Synchronize"/>
<input type="checkbox"/> Use LNS Network Interface	<input type="text"/>
Network Management Mode	<input checked="" type="radio"/> OnNet <input type="radio"/> OffNet
Domain Length (bytes)	6

4. Click **Submit**. A dialog appears informing you that the SmartServer is being switched to standalone mode. It takes approximately 1 minute to switch. When the SmartServer has finished switching to standalone mode, the dialog closes and you can begin using your SmartServer.



See Chapter 5 of the *i.LON SmartServer 2.0 User's Guide* for how to switch the SmartServer from Standalone to LNS mode and synchronize the network attached to the SmartServer to an LNS network database.

If you are using the SmartServer in Standalone mode and your internal FPM devices are using the v40 XIF (your FPM devices have dynamic functional blocks), you should not switch to LNS mode and select an existing LNS network database to be synchronized with your SmartServer. Dynamic functional blocks are not supported in LNS; therefore, the synchronization process may corrupt your LNS network database.

---

## *Adding FPM Devices to the SmartServer*

You can add devices representing your FPM applications to the SmartServer. To do this, you add a new internal device to a LONWORKS channel in the SmartServer tree that uses a static or dynamic interface. If you are integrating your FPM application with another LNS application such as the LonMaker tool, the internal FPM device must use a static interface. If you are running your network with the SmartServer operating as a standalone network manager, the internal device can use a static or dynamic interface.

**Note:** You cannot use the LNS tree to add an internal FPM device.

### *Using a Static Device Interface*

If you are integrating your FPM application with another LNS application such as the LonMaker tool, the internal FPM device must use a static interface. To use a static interface for your FPM, you select a device interface (XIF) file from the root/lonworks/import/<YourCompany> folder on the SmartServer flash disk. This is the XIF file that you generated for your FPM with the *i.LON* LonWorks Interface Developer tool (see Chapter 4 for more information on creating XIF files for FPMs).

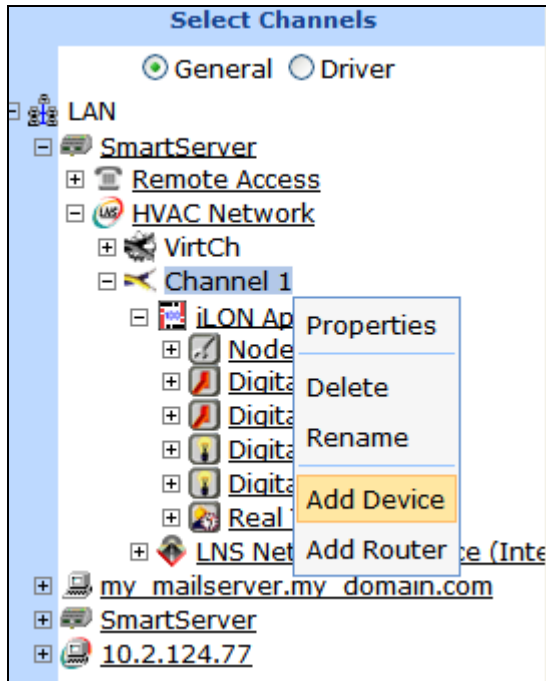
To add an FPM device that uses a static interface to a SmartServer, follow these steps:

1. If you are integrating your FPM with an LNS application such as the LonMaker tool, verify that you have completed the following steps:
  - a. Installed EES 2.0 and LNS Server Service Pack 5 on the *i.LON* SmartServer 2.0 DVD or the *i.LON* SmartServer 2.0 Programming Tools DVD.
  - b. Added an LNS Server to the LAN.
  - c. Configured the SmartServer to use LNS network management services (**LNS Auto** or **LNS Manual**) and synchronized the SmartServer to an LNS network database.

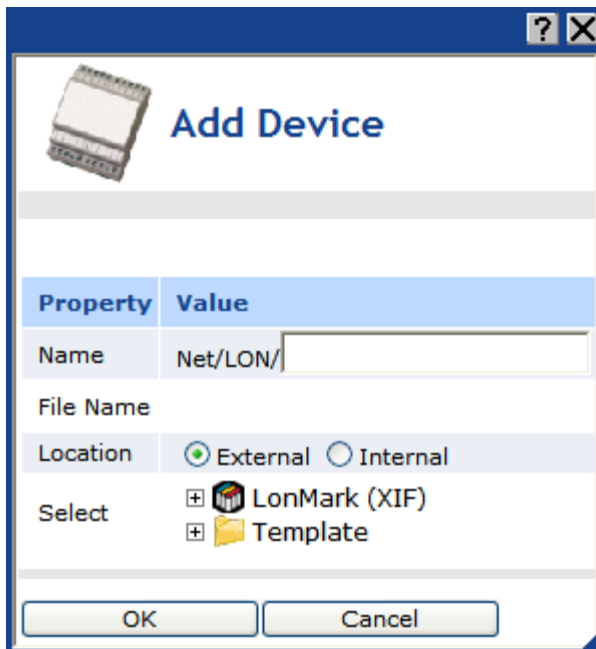
See *Using LNS Network Management Services* earlier in this chapter for how to do complete these steps.

2. Expand the network icon in the SmartServer tree, right-click a LONWORKS channel, and then select **Add Device** on the shortcut menu.

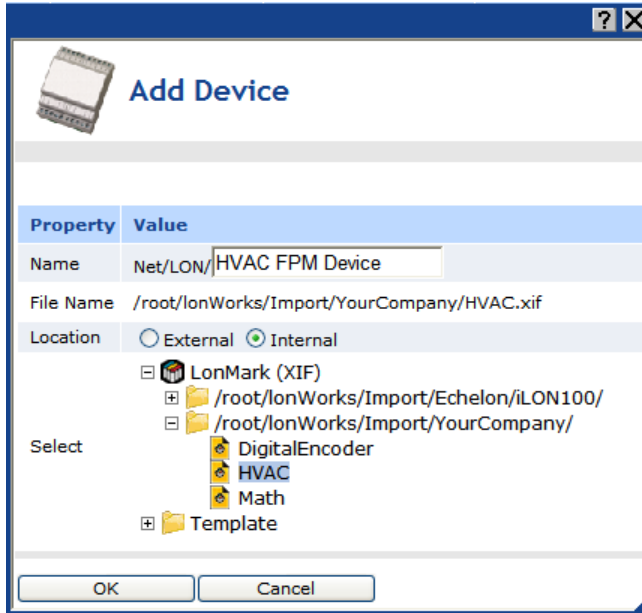




3. The **Create Device** dialog opens.

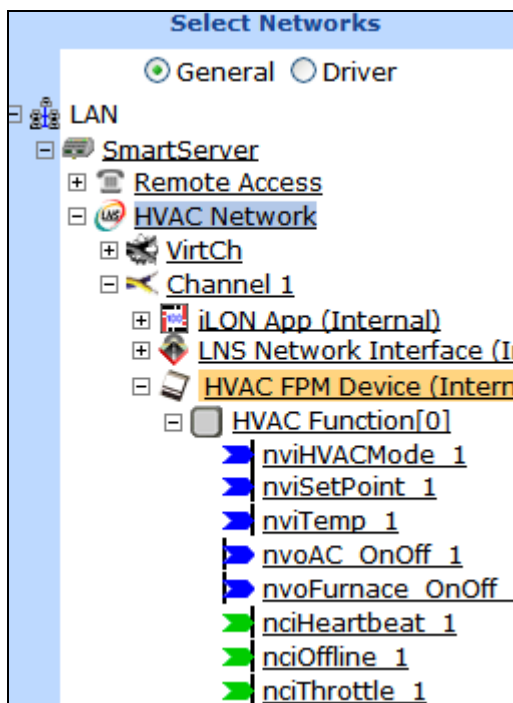


4. In the **Name** property, enter a meaningful name for the device.
5. In the Location property, select **Internal**.
6. Expand the **LonMark (XIF)** folder, expand the root/lonworks/import/<YourCompany> folder, and then select the XIF created for your FPM application.



7. Click **OK**. An internal device representing your FPM application is added to the bottom of the **LON** channel tree.
8. Click **Submit**. You must wait approximately 15 seconds for the SmartServer to instantiate the XIF file used for the internal device. Once the XIF has been instantiated, you can expand the FPM device and its functional block to show the data points in the FPM application.

**Note:** The FPM device will be highlighted orange in the SmartServer tree, indicating that it not commissioned. If you are running your LONWORKS network in LNS mode (**LNS Auto** or **LNS Manual**) and you plan on using LONWORKS connections to bind the data points in your FPM application, you must first commission your FPM device. See *Commissioning FPM Devices* later in this chapter for more information on how to do this. You do not need to commission the FPM device in order for it to run its application.

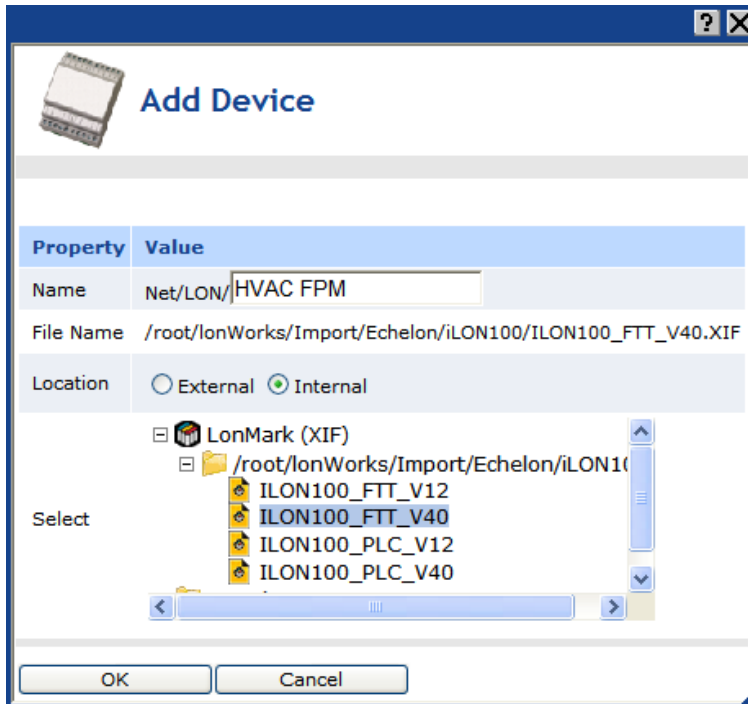


## Using a Dynamic Device Interface

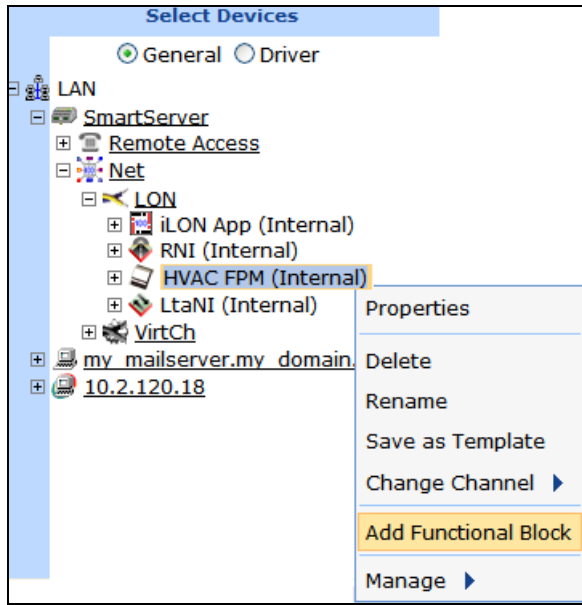
If you are running your network with the SmartServer operating as a standalone network manager, the internal device can use a static or dynamic interface. To use a dynamic interface, you select the SmartServer's v40 XIF from the root/lonworks/import/Echelon/iLON100 folder, and you then add a dynamic functional block to the device that uses a UFPT representing your FPM application.

To add an FPM device that uses a dynamic interface to a SmartServer, follow these steps:

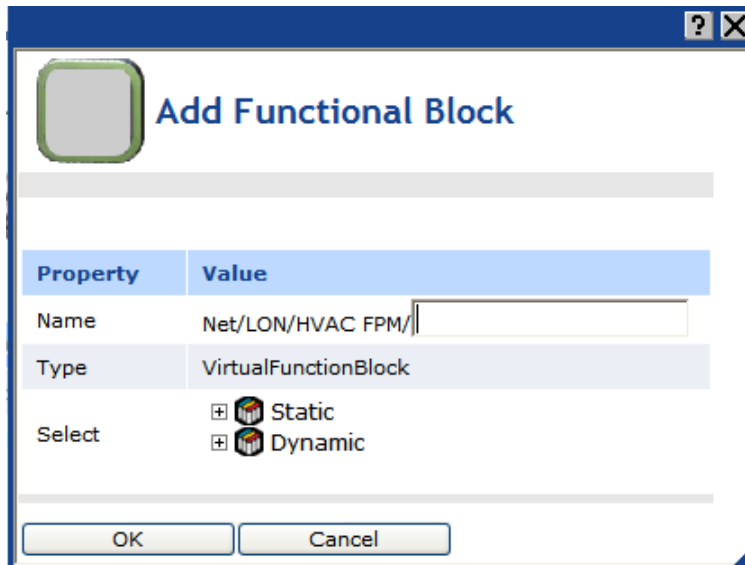
1. Add a new internal device to a LONWORKS channel following steps 2–5 in the previous section, *Using a Static Device Interface*.
2. Expand the **LonMark (XIF)** folder, expand the root/lonworks/import/Echelon/iLON100 folder, and then select the appropriate v40 XIF for your SmartServer (the **ILON\_FTT\_V40** XIF if you have the FTT model of the SmartServer; the **ILON\_PLC\_V40** XIF if you have the PL model).



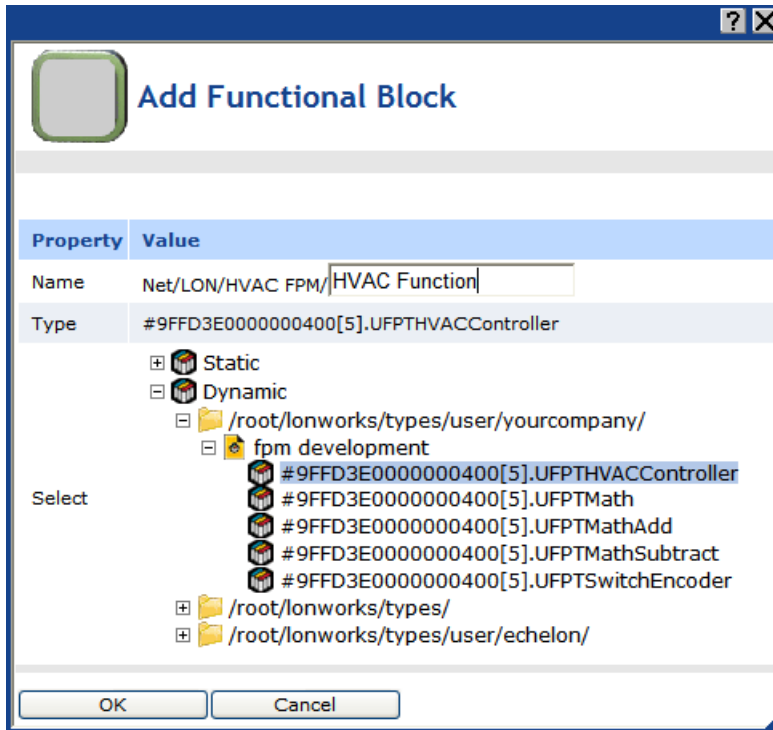
3. Click **OK**. A new device with the name you specified is added to the tree under the **LON** channel.
4. Click **Submit**. You must wait approximately 15 seconds for the SmartServer to instantiate the v40 XIF file used for the internal device before you can add an FPM functional block to the device as described in the following steps.
5. Verify that your company's updated resource file set, which should include the UFPTs on which your FPMs are based, is installed in the root/lonworks/types folder on the SmartServer flash disk. If your updated resource file set is not on a SmartServer, you will not be able to create FPM functional blocks on that SmartServer. See Chapter 3, *Creating FPM Templates*, for more information on how to generate your company's resource file set and copy it to a SmartServer.
6. Right-click the internal FPM device you created, and then select **Add Functional Block** on the shortcut menu.



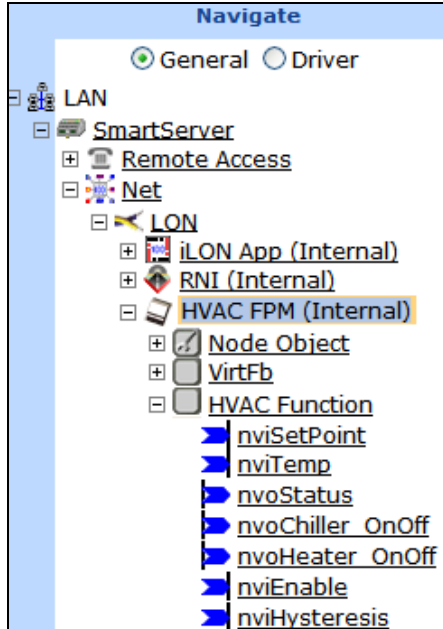
7. The **Add Functional Block** dialog opens.



8. In the **Name** property, enter a name for the functional block that summarizes the functionality of the FPM.
9. In the **Select** property, expand the **Dynamic** folder, expand the folder in the root/lonworks/types/user directory containing your company's resource file set, expand your company's resource file set to show the available UFPTs, and then select the UFPT representing the FPM application (*<company program ID>.UFPT<FPM Name>*).



10. Click **OK**.
11. Click **Submit**. A functional block representing the FPM application and all of the data points declared in the FPM application are added to the tree under the internal device.



**Note:** The FPM device will be highlighted orange in the SmartServer tree, indicating that it not commissioned; however, you do not need to commission the FPM device in order for it to run its application.

## Commissioning FPM Devices

If you are running your LONWORKS network in LNS mode (**LNS Auto** or **LNS Manual**) and you plan on using LONWORKS connections to bind the data points in your FPM application with the data points on the internal SmartServer device, on another FPM device, or on the external devices connected to the SmartServer, you must first commission your FPM device. You can commission your FPM device using the SmartServer tree or LNS tree in the SmartServer Web interface, or using an LNS application such as the LonMaker tool.

Once your FPM device is commissioned in the LonMaker tool, you cannot use the SmartServer to change the **Commission Status** or **Application Status** of the device. You can only use the LonMaker tool to decommission and re-commission the device, and to set the device application online or offline.

### Commissioning FPM Devices with the SmartServer

To commission your FPM device with the SmartServer tree or LNS tree in the SmartServer Web interface, follow these steps:

1. Click **Driver** above the navigation pane in the left side of the SmartServer Web interface.
2. Expand the LNS Server (if in the LNS tree), network, and channel containing the FPM device to be commissioned, and then click the FPM device. The **Setup – LON Device Driver** Web page opens.
3. Select the **Smart Network Management** check box in the **Smart Network Management** column header.

The screenshot shows the 'Setup - LON Device Driver' web interface. On the left, a tree view shows the hierarchy: LAN > SmartServer > Remote Access > HVAC Network > VirtCh > Channel 1 > iLON App (Internal) > LNS Network Interface (Internal) > HVAC FPM Device (Internal) > HVAC Function[0] > nviHVACMode\_1 > nviSetPoint\_1 > nviTemp\_1 > nvoAC\_OnOff\_1 > nvoFurnace\_OnOff\_1 > nciHeartbeat\_1 > nciOffline\_1 > nciThrottle\_1. The 'Smart Network Management' checkbox is checked and highlighted with a red box. The right pane shows the configuration for the selected device: Name: HVAC Network/Channel 1/HVAC FPM Device, Handle: 16, Description: (empty). Below this is a table of properties:

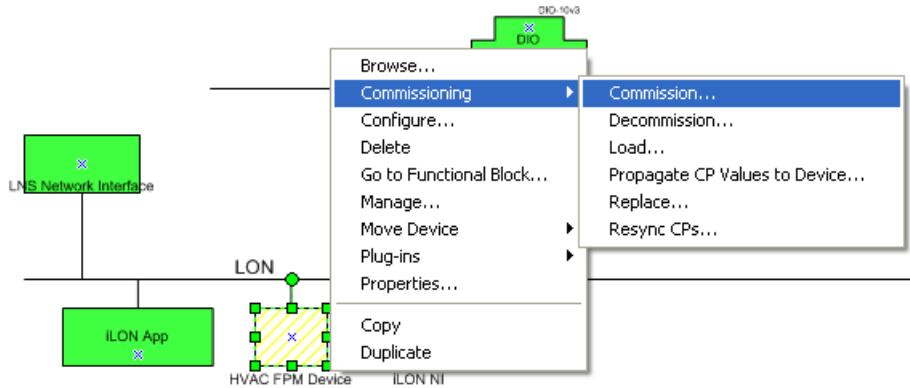
Lon Device Property	Value
Icon	App
Hidden	<input type="checkbox"/>
<input checked="" type="checkbox"/> Smart Network Management	
Progress	
Identification Property	
Neuron ID	820000197686
Requested Program ID	9ffd3e0000000400
Maximum Number of Dynamic Functional Blocks	0
Maximum Number of Dynamic Data Points	0

4. Click **Submit**. The FPM device is commissioned and its corresponding icon in the SmartServer tree should be clear.

### Commissioning FPM Devices with the LonMaker Tool

To commission your FPM device with the LonMaker tool, follow these steps:

1. In the LonMaker tool, right-click the FPM device, point to **Commissioning**, and then click **Commission** in the shortcut menu.

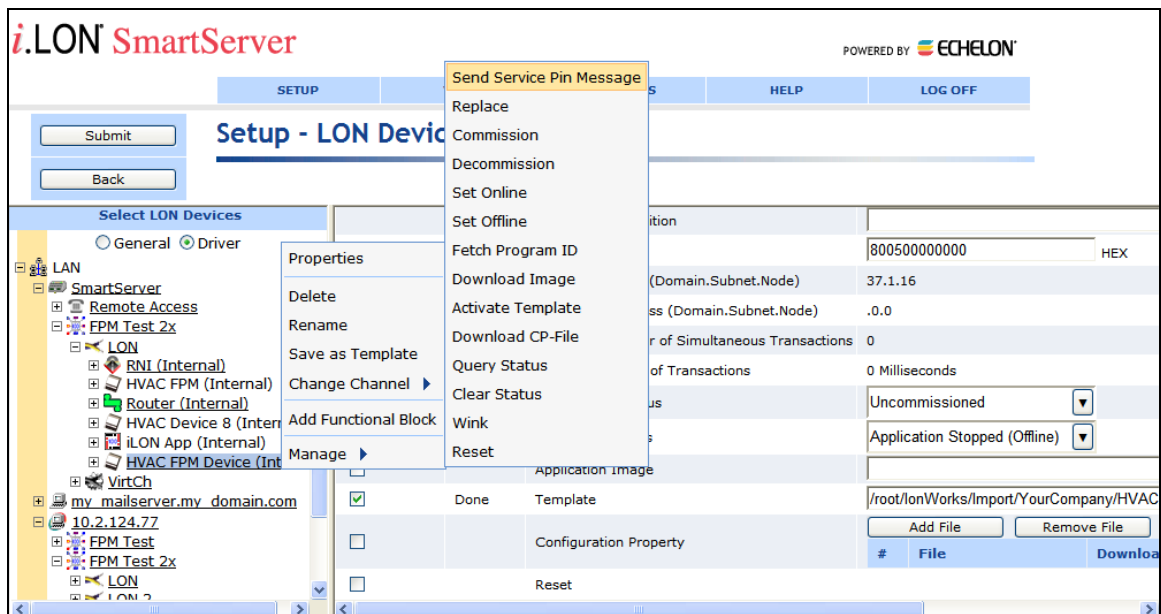


2. Follow the instructions in the Commission Device Wizard and then click **Finish**. See the *LonMaker User's Guide* for more information on using this wizard.
3. When the LonMaker tool is done commissioning the FPM device, the FPM device shape will be solid green (online) or crosshatched green (soft offline), indicating that the FPM device has been commissioned. In addition, the FPM device should be clear in the SmartServer tree.

### Recommissioning FPM Devices

If you decommission the FPM device, you can re-commission it with the LonMaker tool and the SmartServer Web interface following these steps:

1. In the LonMaker tool, right-click the FPM device, point to **Commissioning**, and then click **Commission** in the shortcut menu.
2. Follow the instructions in the Commission Device Wizard and then click **Finish**.
3. When the service pin dialog opens, right-click the FPM device in the SmartServer tree in the navigation pane on the left side of the SmartServer Web interface, point to **Manage**, and then click **Send Service Pin Message** on the shortcut menu.



4. The LonMaker tool recommissions the FPM Device. When the LonMaker tool is done, the FPM device shape will be solid green (online) or crosshatched green (soft offline), indicating that the FPM device has been commissioned

## Testing FPM Applications

After you add an FPM application on the SmartServer, you can test it using the **View – Data Points** Web page. To do this, you open the **View – Data Points** Web page, add the input and output data points in the FPM device that you can use to observe the FPM application processing data point updates, update one or more of the input data points, and observe that the output data points are updated accordingly.

To test an FPM on your SmartServer, follow these steps:

1. Click **View** and then click **Data Points**. The **View – Data Points** Web page opens.
2. Close the graph by clicking the 'X' in the upper right-hand corner of the application frame.
3. Under the FPM functional block, click the input data points that affect the values stored in the output data points in the same FPM functional block and then add the output data points. The data points appear in the **View – Data Points** Web page.

**View - Data Points**

Submit Back

Select Data Point: General Driver

LAN

- SmartServer
  - Remote Access
  - Net
    - LON
      - iLON App (Internal)
      - RNI (Internal)
      - HVAC FPM (Internal)
        - Node Object
        - VirtFb
        - HVAC Function
          - nviSetPoint
          - nviTemp
          - nvoStatus
          - nvoChiller\_OnOff
          - nvoHeater\_OnOff
          - nviEnable
          - nviHysteresis

Show Graph

	Name	Format	Value	Unit	Priority	Status
0	Net/LON/HVAC FPM/HVAC Function/nviTemp	SNVT_temp_p	0.00	degrees C	255	NUL
1	Net/LON/HVAC FPM/HVAC Function/nviSetPoint	SNVT_temp_p	0.00	degrees C	255	NUL
2	Net/LON/HVAC FPM/HVAC Function/nviHysteresis	SNVT_temp_p	0.00	degrees C	255	NUL
3	Net/LON/HVAC FPM/HVAC Function/nvoChiller_OnOff	SNVT_switch	OFF	---	255	NUL

4. Enter values for the input data points that will cause the FPM application to update the value stored in the output data point.

**View - Data Points**

Show Graph

	Name	Format	Value	Unit	Priority	Status
0	Net/LON/HVAC FPM/HVAC Function/nviTemp	SNVT_temp_p	78	degrees C	255	ONLINE
1	Net/LON/HVAC FPM/HVAC Function/nviSetPoint	SNVT_temp_p	72.5	degrees C	255	ONLINE
2	Net/LON/HVAC FPM/HVAC Function/nviHysteresis	SNVT_temp_p	4.5	degrees C	255	ONLINE
3	Net/LON/HVAC FPM/HVAC Function/nvoChiller_OnOff	SNVT_switch	ON	---	255	ONLINE

5. Observe that output data points are updated accordingly based on the algorithm you wrote in the FPM application. In this example, the nvoChiller\_OnOff output data point (SNVT\_switch) turns on when the nviTemp input data point is greater than the sum of the nviSetPoint and the nviHysteresis input data points (all **SNVT\_temp\_p** types).

## Connecting FPM Data Points

After you verify that your FPM is functioning properly, you can use LONWORKS or Web connections to connect the data points declared in your FPM to the data points on the internal SmartServer device, the data points in another FPM application, and the data points of the external devices connected to the SmartServer.



The major difference between LONWORKS connections and Web connections is that LONWORKS connections propagate data point updates over a LONWORKS channel via the LonTalk Protocol or the LonTalk protocol tunneled through an IP-852 channel. Web connections propagate data point updates via SOAP/HTTP over a TCP/IP network. Web connections provide an alternative solution to LONWORKS connections over an IP-852 channel for connecting devices over multiple networks; however, Web connections are much slower (40 data point updates per second) than LONWORKS IP-852 connections (1,000 updates per second).

To integrate your FPM applications with external devices stored in an LNS network database, you can use LONWORKS connections, which you can create with the LNS tree or an LNS application such as the LonMaker tool. Alternatively, you can copy the data points from the LNS tree to the SmartServer tree via the LNS Proxy Web service and then create Web connections between the data points from the SmartServer tree.

### *Creating LONWORKS Connections*

You can connect the data points in your FPM application using LONWORKS connections. You can create LONWORKS connections using the LNS tree in the SmartServer Web interface (via the LNS Proxy Web service) or using an LNS application such as the LonMaker tool. You can create two types of LONWORKS connections:

- Output data points on the SmartServer or external devices (the source data points) to the input data points declared in the FPM application (the target data points).
- Output data points declared in the FPM application (the source data points) to the input data points on the SmartServer, input data points in another FPM application, or the input data points on external devices connected to the SmartServer (the target data points).

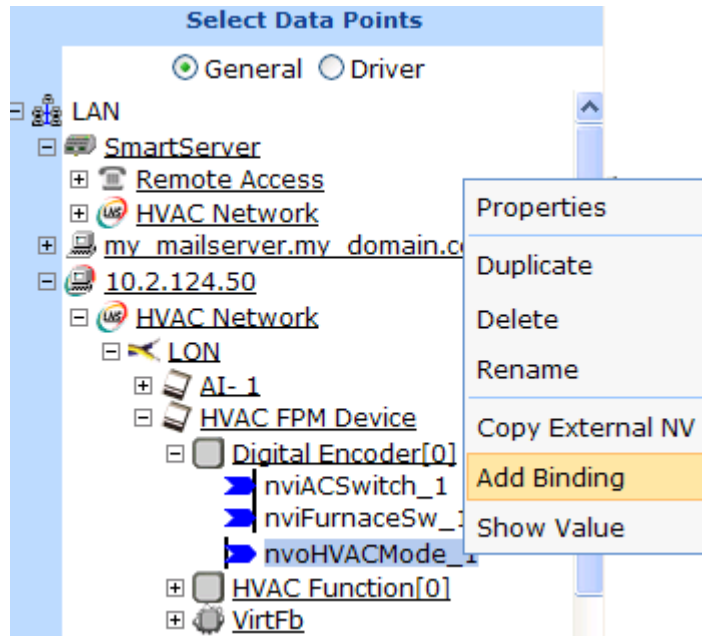
**Note:** If you are using the SmartServer in Standalone mode or your FPM devices are using the v40 XIF, you cannot create LONWORKS connections with the LonMaker tool. In this case, you can create Web connections from the SmartServer tree to connect the data points in your FPM applications. See the next section, *Creating Web Connections*, for how to do this.

### **Connecting FPM Data Points with the LNS Tree**

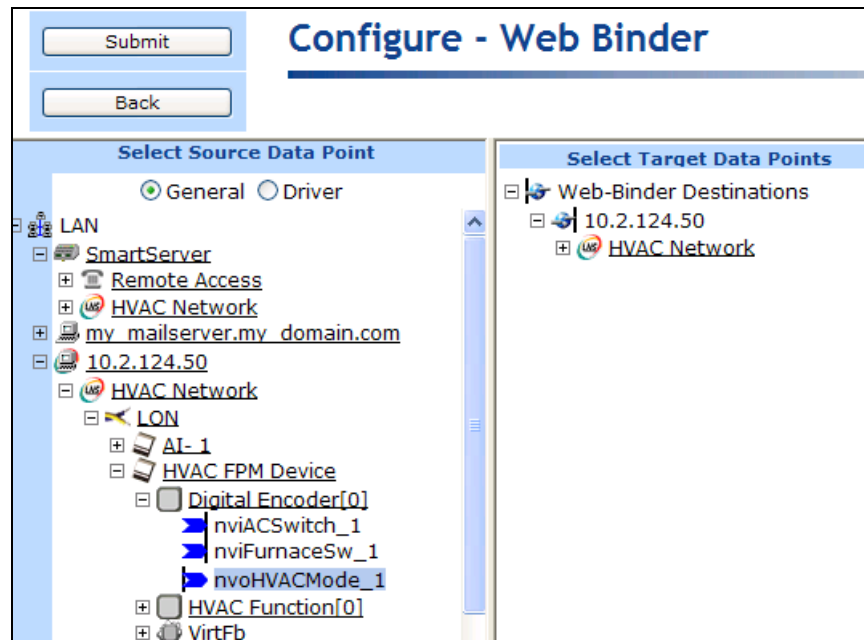
You can create LONWORKS connections with the data points declared in your FPM application from the LNS tree. To do this, follow these steps:

1. Verify that you have completed the following steps:
  - a. Installed EES 2.0 and LNS Server Service Pack 5 on the *i.LON SmartServer 2.0 DVD* or the *i.LON SmartServer 2.0 Programming Tools DVD*.
  - b. Added an LNS Server to the LAN.
  - c. Configured the SmartServer to use LNS network management services (**LNS Auto** or **LNS Manual**) and synchronized the SmartServer to an LNS network database.

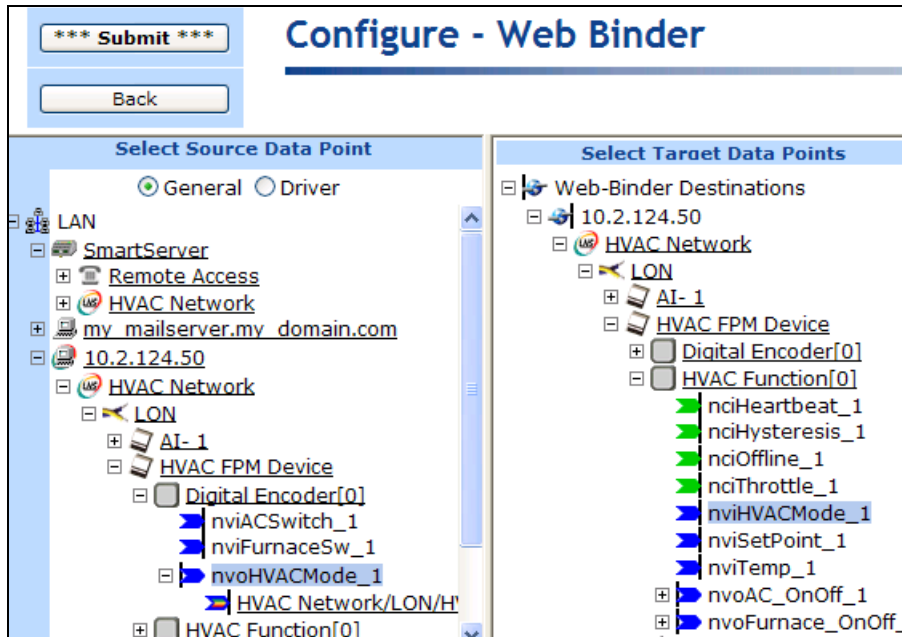
See *Using LNS Network Management Services* earlier in this chapter for how to do complete these steps.
2. Verify that you have commissioned the FPM device using the SmartServer or an LNS application such as the LonMaker tool. See *Commissioning FPM Devices* earlier in this chapter for how to do this.
3. From the LNS tree in the left frame of the SmartServer Web interface, expand the LNS Server, LNS network database, channel, device, and functional block containing the hub (source) network variable.
4. Right-click a hub (source) network variable and then click **Add Binding** in the shortcut menu.




5. The **Configure – WebBinder** Web page opens and the hostname of the LNS Server and the LNS network database in which the hub network variable is stored appear under the WebBinder Destinations icon in the application frame to the right.



6. From the WebBinder Destinations tree on the right frame, expand the LNS network database, expand the network, channel, device and functional block containing the desired target network variables to be connected, and then click one or more compatible target network variables.



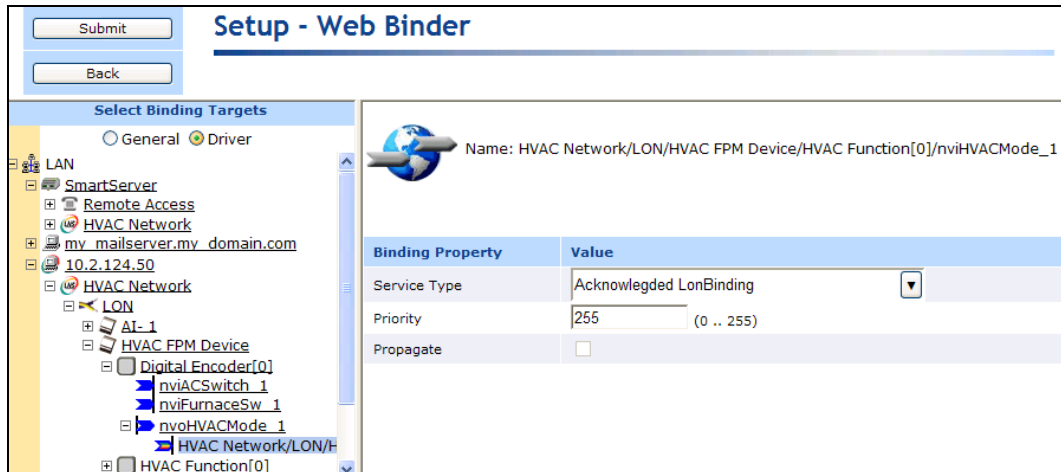
- References to the target LONWORKS network variables (  ) are added underneath the hub network variables in the LNS tree in the left frame. Updates to the selected hub network variable will be propagated to the target network variables listed underneath the hub.

Repeat this step to connect the selected hub network variable to any other desired compatible target hub network variables.

- If the target network variable is not compatible with the hub network variable a warning message appears. You can delete the connection by right-clicking the reference to the target network variable on the LNS tree in the left frame and clicking **Delete** on the shortcut menu. See Chapter 5 of the *i.LON SmartServer 2.0 User's Guide* for more information on how to do this.
  - You can also check whether a LONWORKS CONNECTION is valid by right-clicking the reference to the target network variable on the LNS tree in the left frame and clicking **Validate** on the shortcut menu. The **WebBinder Validation Results** dialog opens and displays the results. See Chapter 5 of the *i.LON SmartServer 2.0 User's Guide* for more information on how to do this.
- Click **Submit**. When an event-driven update defined in the device application occurs, the hub network variable sends an updated value to the selected target network variables.
  - Optionally, you can change the messaging service used for the connection (Acknowledged, Repeating, or Unacknowledged). To do this, click **Driver**, and then select one or more of the target network variables under the hub network variable in the LNS tree.

Note that all LonWorks connections created in the LNS tree use **Subnet/Node ID** addressing. This means that a message packet travels from the sending device to the destination device using the 2-byte logical address of the destination device in the network.

The default messaging service for LonWorks connections created in the LNS tree is **Acknowledged LonBinding**.



See Chapter 5 of the *i.LON SmartServer 2.0 User's Guide* for more information on selecting a messaging service.

10. You can add the hub and target network variables to the **View – Data Points** Web page and test that the LonWorks connections are updating the target network variables accordingly. To test your LONWORKS connections in the LNS tree, follow these steps:
  - a. Click **View** and then click **Data Points**. The **View – Data Points** Web page opens.
  - b. Close the graph by clicking the 'X' in the upper right-hand corner of the application frame.
  - c. In the LNS tree, click the output data points on the SmartServer, an FPM, or on an external device that are bound to input data points in the FPM application. Observe that the input data points in the FPM application have the same values as the output data points to which they are connected.

**View - Data Points**

[Show Graph](#)

---

First Log Entry: 2008-02-01 15:22:20 Last Log Entry: 2008-02-01 15:23:01

	Name	IP Address	Format	Value	Unit	Priority	Status
0	HVAC Network/LON/HVAC FPM Device/Digital Encoder [0]/nvoHVACMode_1	10.2.124.50	SNVT_hvac_mode	HVAC_COOL	HVAC mode names	255	ONLINE
1	HVAC Network/LON/HVAC FPM Device/HVAC Function [0]/nviHVACMode_1	10.2.124.50	SNVT_hvac_mode	HVAC_COOL	HVAC mode names	255	ONLINE

- d. Click the output data points in the FPM application that are bound to the data points on the SmartServer, another FPM, or an external device. Observe that the input data points on the SmartServer or external device have the same values as the output data points in the FPM application to which they are connected.

**View - Data Points**

[Show Graph](#)

---

First Log Entry: 2008-02-01 15:25:33 Last Log Entry: 2008-02-01 15:26:20

	Name	IP Address	Format	Value	Unit	Priority	Status
0	HVAC Network/LON/HVAC FPM Device/HVAC Function[0]/nvoAC_OnOff_1	10.2.124.50	SNVT_switch	0.0 0		255	ONLINE
1	HVAC Network/LON/iLON App/AC State/nviClavalue_1	10.2.124.50	SNVT_switch	0.0 0		255	ONLINE

**Note:** For more information on creating LONWORKS connections with the LNS tree, including how to validate and delete them, see Chapter 5 of the *i.LON SmartServer 2.0 User's Guide*.

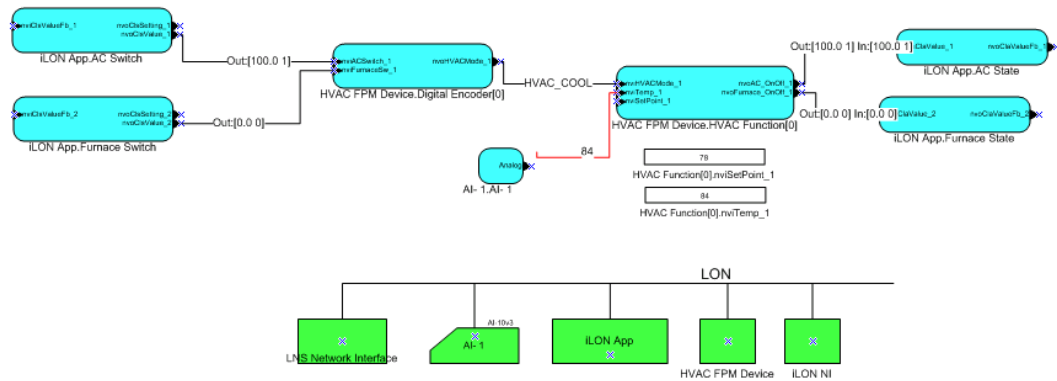
## Connecting FPM Data Points with the LonMaker Tool

You can use the LonMaker tool to create LONWORKS connections with the data points declared in your FPM application. To do this, follow these steps:

1. Verify that you have completed the following steps:
  - a. Installed Echelon Enterprise Services and LNS Server/Turbo Edition SP4 (required if you installed Echelon Enterprise Services SR2) on the *i.LON SmartServer 2.0 DVD* or the *i.LON SmartServer 2.0 Programming Tools DVD*.
  - b. Added an LNS Server to the LAN.
  - c. Configured the SmartServer to use LNS network management services (**LNS Auto** or **LNS Manual**) and synchronized the SmartServer to an LNS network database.

See *Using LNS Network Management Services* earlier in this chapter for how to do complete these steps.

2. Verify that you have commissioned the FPM device using the SmartServer or an LNS application such as the LonMaker tool. See *Commissioning FPM Devices* earlier in this chapter for how to do this.
3. Connect the data points in your FPM application using either the Connector shape in the LonMaker Basic Shapes stencil, the **Connector** tool on the Visio **Standard** toolbar, or the **Network Variable Connection** dialog box. See the *LonMaker User's Guide* for more information on creating LONWORKS connections with these methods.
4. Monitor the LONWORKS connections to observe that the data points in the FPM application and the data points on the devices to which the FPM data points are bound are being updated accordingly.



## Creating Web Connections

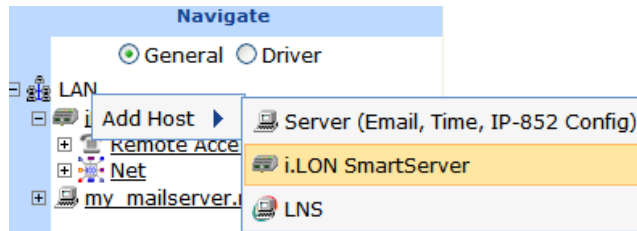
You can connect the data points in your FPM application using Web connections. You can use Web connections if you want to use polled updates to process data point values, or you can use them if you are running the network with the SmartServer operating in **Standalone** mode (LONWORKS connections are not supported in this mode). Typically, you will create two types of Web connections:

- Output data points on the SmartServer or external devices (the source data points) to the input data points declared in the FPM application (the target data points).
- Output data points declared in the FPM application (the source data points) to the input data points on the SmartServer, input data points in another FPM application, or the input data points on external devices connected to the SmartServer (the target data points).

To create Web connections with the data points declared in your FPM application, follow these steps:

1. If you want to use Web connection to bind the data points in your FPM application to data points on SmartServers other than your local SmartServer, you can add one or more remote SmartServers to the LAN. To add a remote SmartServer to the LAN, follow these steps:

- a. Right-click the **LAN** icon, point to **Add Host**, and then click **i.LON SmartServer 2.0** on the shortcut menu.



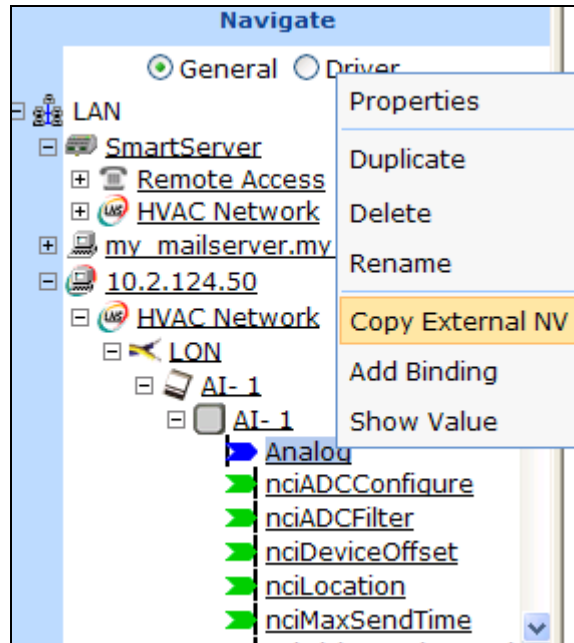
- b. The **Setup – Remote i.LON SmartServer 2.0** Web page opens, and a SmartServer icon is added to the tree view below the **LAN** icon.

The image shows a web page titled 'Setup - Remote i.LON SmartServer'. It features a server icon and a text field for 'IP or Hostname' with the value '0.0.0.0'. Below this is a table for 'Host Property' and 'Value'.

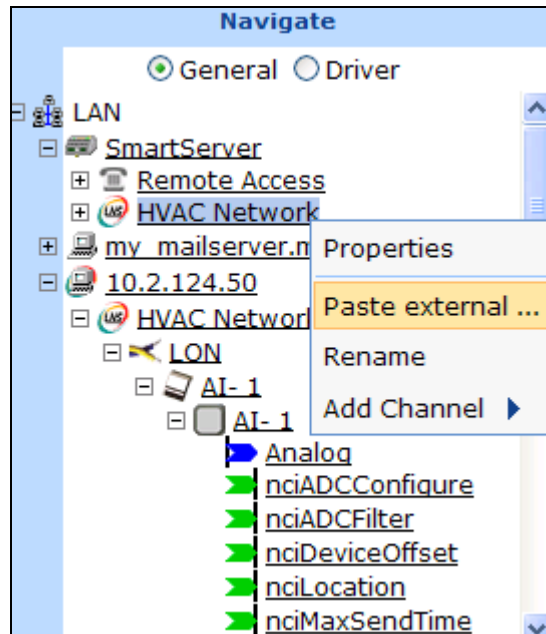
Host Property	Value
SOAP Path	/WSDL/iLON100.WSDL
HTTP Port (Web Server / SOAP)	80
Retry Time (defaults to 120 s)	120 Seconds
SOAP User Name *	
SOAP Password *	Change Password
Format values in WebBinder SOAP messages using	Data Point Format

\* For i.LON SmartServer Destination Servers, SOAP Authentication Parameters may be configured in the webparams.dat file

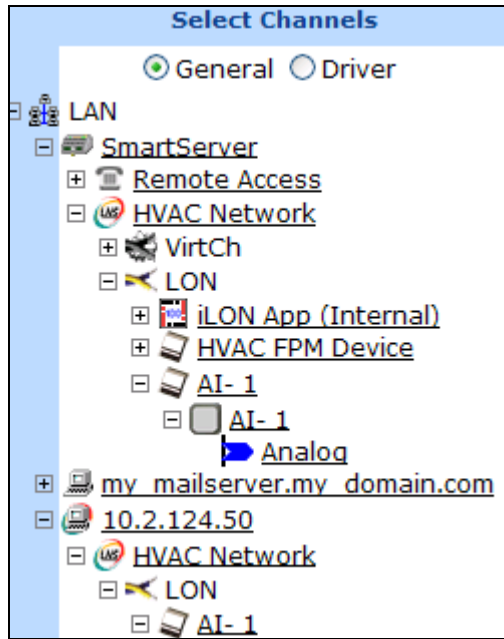
- a. Configure the SOAP/HTTP properties of the SmartServer. See Chapter 3 of the *i.LON SmartServer 2.0 User's Guide* for more information on configuring these properties.
  - d. Click **Submit**.
2. If you want to use Web connections to bind the data points in your FPM application to the data points of external devices stored in an LNS network database, you can copy the data points on the external devices from the LNS tree to the SmartServer tree via the LNS Proxy Web service. To do this, follow these steps:
    - a. Follow the steps in *Selecting LNS Network Management Services* to install the Echelon i.LON SmartServer 2.0 Enterprise Services, add an LNS Server to the LAN, and configure the SmartServer to use LNS network management services (**LNS Auto** or **LNS Manual**).
    - b. In the LNS tree, expand the LNS network database, channel, device, and functional block containing the network variable or configuration property to be added to the SmartServer tree, right-click the network variable or configuration property, and then select **Copy External NV** on the shortcut menu. To copy multiple network variables or configuration properties, click one, and then either hold down CTRL and click all others to be copied or hold down SHIFT and select another to select the entire range.



- c. In the tree of the target SmartServer, right-click the any object in the network branch and click **Paste External...** on the shortcut menu.

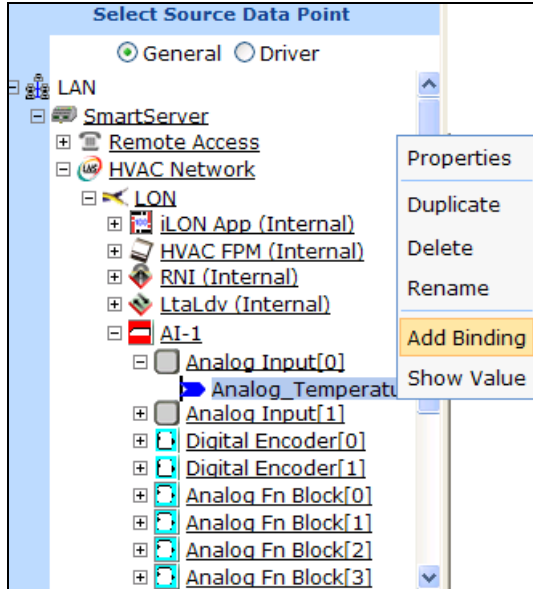


- d. The data points and their parent channel, device, and functional block are added to the network tree of the target SmartServer.



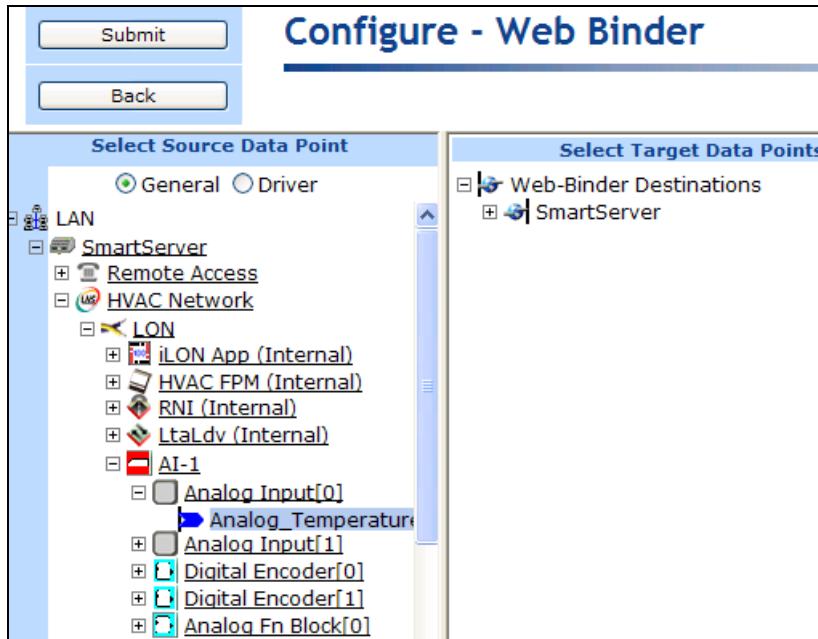
e. Click **Submit**.

- From the local SmartServer tree on the left frame, right-click a source data point and then click **Add Binding** in the shortcut menu. The source data point will typically be an output data point on the internal SmartServer device, an output data point on an external device connected to the SmartServer, or an FPM output data point. Updates to the source data point in a Web connection are propagated to one or more target data points.



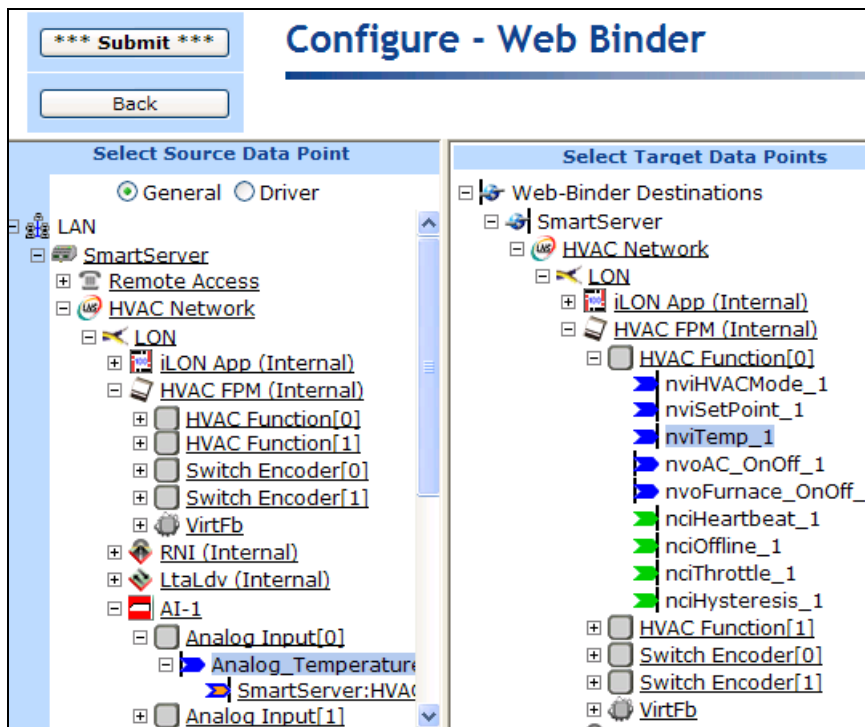
- The **Configure – Web Binder** Web page opens and the hostnames of the local SmartServer and any remote SmartServers added to the LAN, which are collectively referred to as *Webbinder Destinations*, appear in the application frame to the right. If a Webbinder Destination cannot be reached, a single child node called “Target” appears with the IP address of the SmartServer below the Webbinder Destinations icon.





- From the WebBinder Destinations tree on the right frame, expand the SmartServer WebBinder destination icon containing the target data points to be connected, expand the network, channel, device, and functional block containing the desired target data point, and then click one or more compatible target data points.

The target data point will typically be an input data point declared in the FPM application (if the source data point is an output data point on the internal SmartServer device or an output data point on an external device), or it will be an input data point on the internal SmartServer device or an external device connected to the SmartServer (if the source data point is an FPM output data point).



6. Click **Submit**. References to the target data points are added underneath the source data point in the local tree on the left frame.
7. Test that your Web connections are updating the FPM data points accordingly following these steps:
  - a. Click **View** and then click **Data Points**. The **View – Data Points** Web page opens.
  - b. Close the graph by clicking the ‘X’ in the upper right-hand corner of the application frame.
  - c. In the tree, click the data points under the FPM functional block that are members of a Web connection. Click the data points on the SmartServer that are bound to the FPM data points in the Web connections.
  - d. In the following example, observe that the **nvoHVACMode\_1** output data point on the HVAC device (#2) has the same value as the **nviHVACMode\_1** input data point on a different functional block on the HVAC device (#3); the **Analog\_Temperature** data point on the external Analog Input device (#4) has the same value as the **nviTemp\_1** input data point on the HVAC FPM device (#5); and the **nvoAC\_OnOff\_1** output data point in the HVAC FPM (#8) has the same value as the **nviClavalue1** data point on the SmartServer (#9). These data points are bound with Web connections, which keep these data points synchronized.

**View - Data Points**

[Show Graph](#)

First Log Entry: 2008-02-03 16:51:16 Entire Range Last Log Entry: 2008-02-03 16:53:50

	Name	Format	Value	Unit	Priority	Status
0	HVAC Network/LON/iLON App/Digital Input 1/nviClavalueFb_1	SNVT_switch.state	1		255	ONLINE
1	HVAC Network/LON/HVAC FPM/Switch Encoder [0]/nviACSwitch_1	SNVT_switch.state	1	state code	255	ONLINE
2	HVAC Network/LON/HVAC FPM/Switch Encoder [0]/nvoHVACMode_1	SNVT_hvac_mode	HVAC_COOL	HVAC mode names	255	ONLINE
3	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviHVACMode_1	SNVT_hvac_mode	HVAC_COOL	HVAC mode names	255	ONLINE
4	HVAC Network/LON/AI-1/Analog Input [0]/Analog_Temperature	SNVT_temp_f#US	78	°F	255	ONLINE
5	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviTemp_1	SNVT_temp_f#US	78	°F	255	ONLINE
6	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviSetPoint_1	SNVT_temp_f#US	72.5	°F	255	ONLINE
7	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nciHysteresis_1	UCPTHysteresis	4.5	°F	255	ONLINE
8	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nvoAC_OnOff_1	SNVT_switch	ON		255	ONLINE
9	HVAC Network/LON/iLON App/Digital Output 1/nviClavalue_1	SNVT_switch	ON		255	ONLINE

- e. Enter different values for the input data points in the Web connections and observe that the FPM application processes the updated values received by its input data point and writes a value to the output point, The Web connection propagates the updated value in the FPM output data point to the input point to which it is connected, which may be on the SmartServer, on another FPM, or on an external device.

In this example, the value in the **Analog\_Temperature** data point on the external Analog Input device (#4) has dropped to 65°F. The Web connection propagates this value to the **nviTemp\_1** input data point on the HVAC FPM device to which it is connected (#5). The decreased temperature is calculated by the FPM application, which results in its **nvoAC\_OnOff\_1** data point (#8) being changed to OFF. The Web connection then propagates the updated OFF value in the **nvoAC\_OnOff\_1** data point to the **nviClavalue1** data point on the SmartServer (#9).

View - Data Points						
<a href="#">Show Graph</a>						
First Log Entry: 2008-02-03 16:51:16 <span style="float: right;">Entire Range</span> Last Log Entry: 2008-02-03 16:54:42						
	Name	Format	Value	Unit	Priority	Status
0	HVAC Network/LON/iLON App/Digital Input 1/nviClisValueFb_1	SNVT_switch.state	1		255	ONLINE
1	HVAC Network/LON/HVAC FPM/Switch Encoder [0]/nviACSwitch_1	SNVT_switch.state	1	state code	255	ONLINE
2	HVAC Network/LON/HVAC FPM/Switch Encoder [0]/nvoHVACMode_1	SNVT_hvac.mode	HVAC_COOL	HVAC mode names	255	ONLINE
3	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviHVACMode_1	SNVT_hvac.mode	HVAC_COOL	HVAC mode names	255	ONLINE
4	HVAC Network/LON/AI-1/Analog Input [0]/Analog_Temperature	SNVT_temp_f#US	65	°F	255	ONLINE
5	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviTemp_1	SNVT_temp_f#US	65	°F	255	ONLINE
6	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviSetPoint_1	SNVT_temp_f#US	72.5	°F	255	ONLINE
7	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nviHysteresis_1	UCPTHysteresis	4.5	°F	255	ONLINE
8	HVAC Network/LON/HVAC FPM/HVAC Function [0]/nvoAC_OnOff_1	SNVT_switch	OFF		255	ONLINE
9	HVAC Network/LON/iLON App/Digital Output 1/nviClasValue_1	SNVT_switch	OFF		255	ONLINE

**Note:** For more information on using Web connections, including how to validate, delete, and add attachments to them, see Chapter 4 of the *i.LON SmartServer 2.0 User's Guide*.

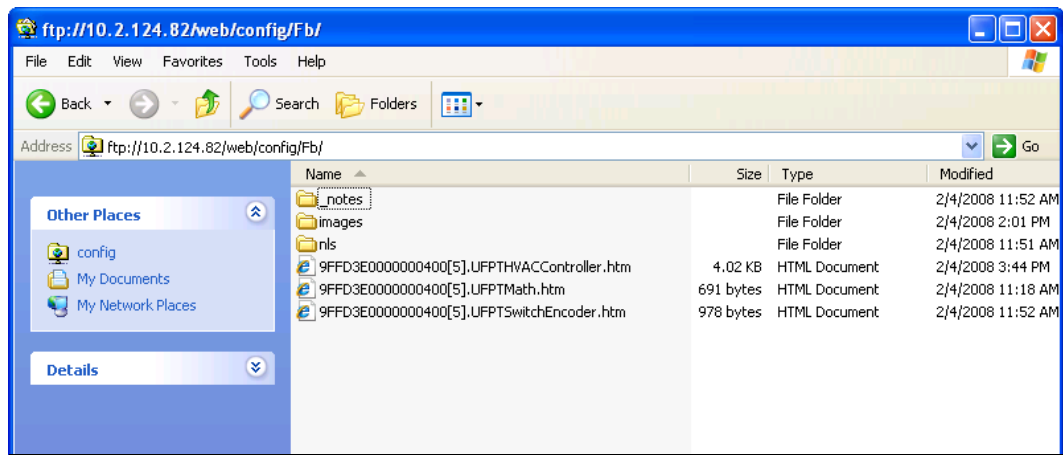
## Creating Custom FPM Configuration Web Pages

You can create configuration Web pages for your FPM applications using *i.LON Vision 2.0*. To do this, you add *i.LON Vision 2.0* read/write and application objects to the default custom Web page that was created for your FPM when you uploaded it to the SmartServer flash disk. The default custom Web page for your FPM is located in the **root/web/config/FB** folder on the SmartServer flash disk.

Once you publish the FPM configuration Web page, you can click the **General** button above the navigation pane on the left side of the SmartServer Web interface, click the functional block representing your FPM application, and use the configuration Web page to read and write values to the data points in your FPM application. In addition, all instances of the same functional block in a static device interface and any new FPM devices you create will automatically have this custom FPM configuration page built for them.

To create a custom FPM configuration Web page follow these steps:

1. Verify that a default configuration Web page has been created for your FPM application. To do this, use an FTP client to browse to the root/web/config/Fb/folder on the SmartServer flash disk and confirm that an **.htm** file with the program ID and name of your FPM application is in the folder.

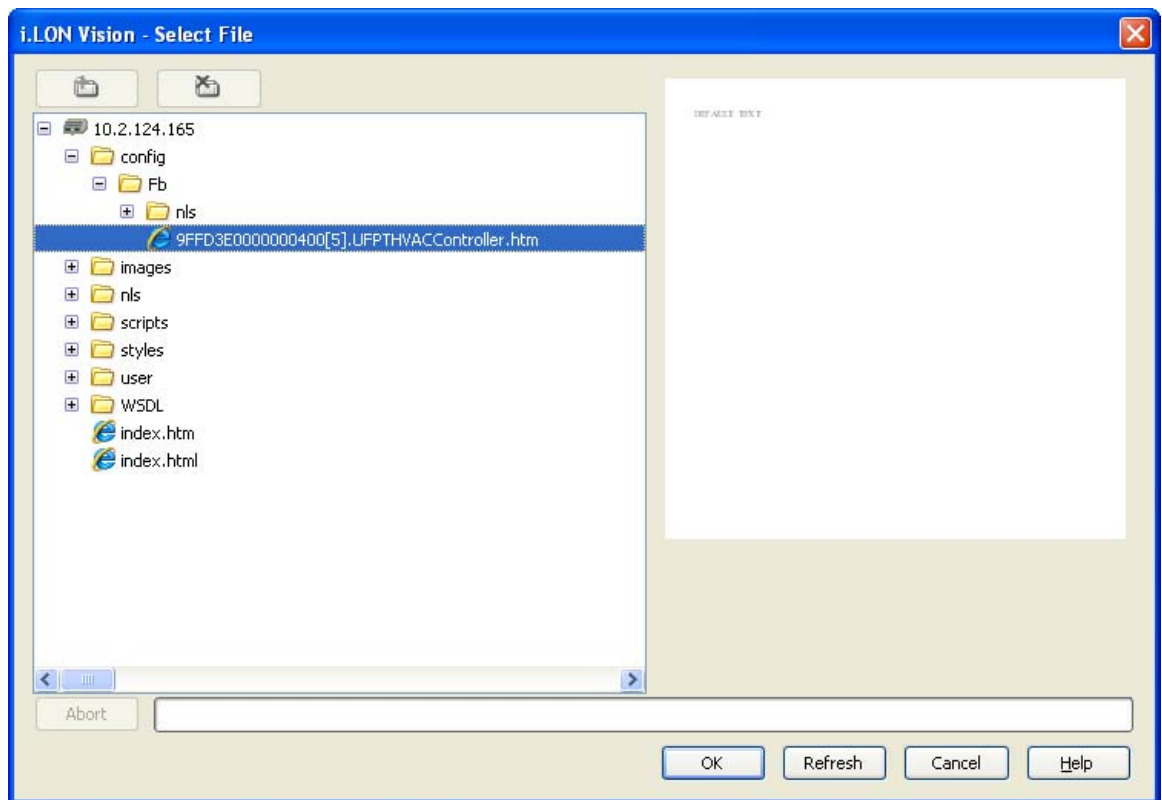


If an **.htm** file for your FPM application is not in the folder, use the *i.LON* Development tool to create the default FPM configuration Web page following these steps:

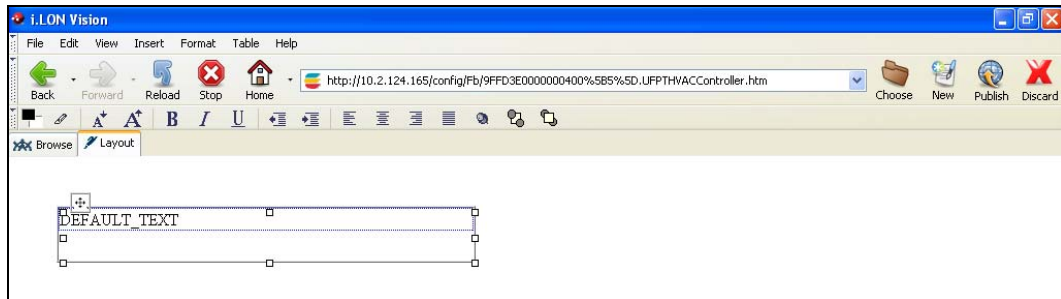
- a. In the **C/C++ Projects** view, expand the **Release** folder, right-click the *<company program ID>.UFPT<FPM name>.app* file and then click **Transfer to i.LON SmartServer** in the shortcut menu.
  - b. In the Deployment Settings window of the **Install FPM Module** dialog, select the **Default Web Page** check box.
  - c. Click **Finish**. The FPM executable module is reloaded on your SmartServer and the custom FPM configuration Web page for the FPM application is created.
2. Install the *i.LON Vision 2.0* software from the *i.LON SmartServer 2.0 DVD* or the *i.LON SmartServer 2.0 Programming Tools DVD*, and then create a Website connection between *i.LON Vision 2.0* and your SmartServer. For more information on how to do this, see the *i.LON Vision 2.0 User's Guide*.



3. Click **Choose** on the Editor toolbar (Choose). The **Select File** dialog opens.
4. Browse the root/web/config/Fb directory on the SmartServer flash disk, select the **.htm** file for your FPM configuration Web page, and then click **OK**.



5. Click **Edit** on the Editor toolbar (Edit) or click the **Layout** tab.
6. Observe the layer containing the “DEFAULT\_TEXT” string. This is an **NLS Text** object that was automatically added to your Web page when it was created.



The **NLS Text** object is used to translate custom SmartServer Web pages into multiple languages. The **NLS Text** object provides a single user-defined key that you can associate with multiple text strings in different languages. These text strings are saved in **.properties** files corresponding to the custom Web page and their respective languages (e.g., `page.properties`, `page_de.properties`, `page_es.properties`, and so on).

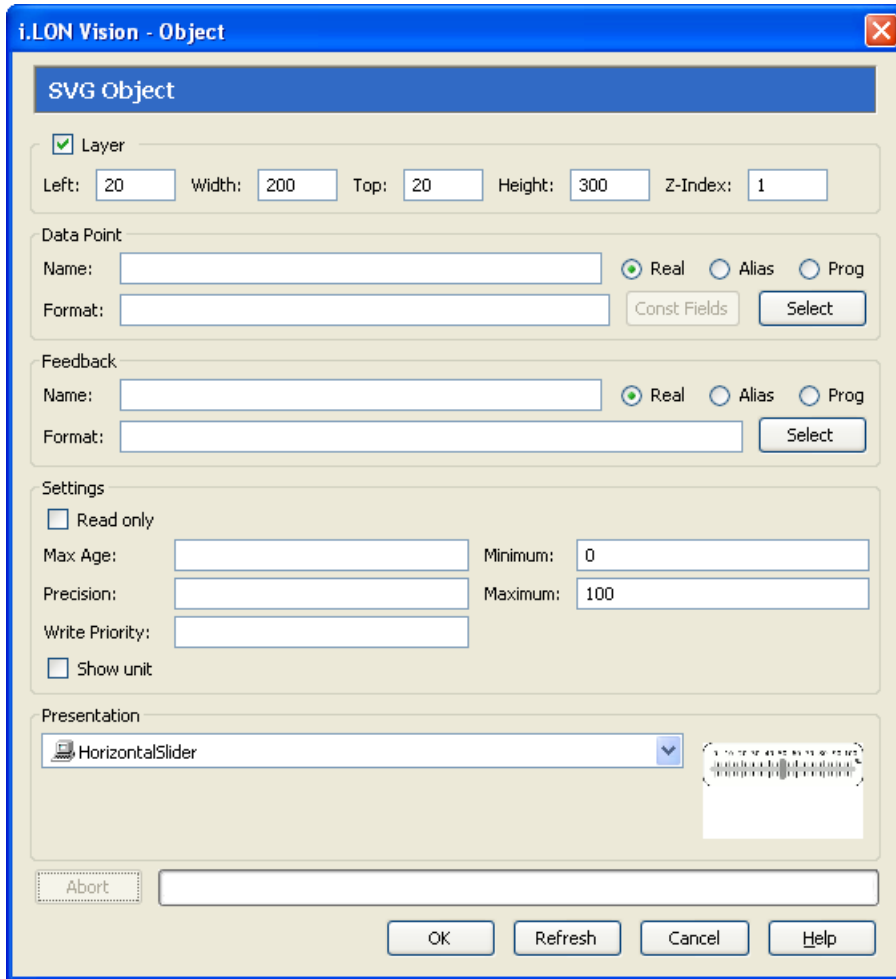
The **.properties** files are stored in the **nls** folder in the same directory as your custom FPM configuration Web page. An **nls** folder containing a default **.properties** file was automatically added to the `web/config/fb` directory on your SmartServer when your custom FPM configuration Web page was created. You can edit the **.properties** file and create more for other languages using either the demo or full version of the *i.LON SmartServer 2.0 Programming Tool*. See *Creating Localized FPM Configuration Web Pages* in Chapter 8 for more information on using the *i.LON SmartServer 2.0 Programming Tool* to translate the **NLS Text** objects in your custom FPM configuration Web pages.

You can edit or delete the provided “`DEFAULT_TEXT`” **NLS Text** object, and you can add additional **NLS Text** objects to your custom FPM configuration Web page. See the *i.LON Vision 2.0 User’s Guide* for more information on the **NLS Text** object.

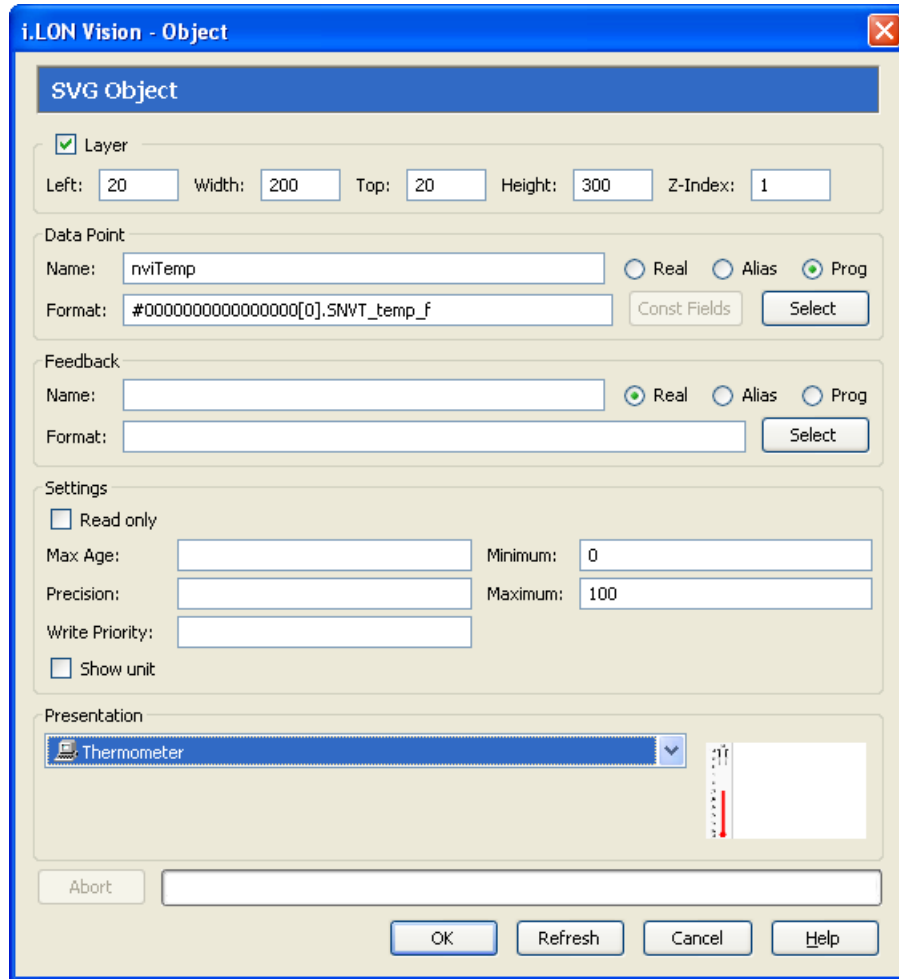
7. Click **Insert** and then select an *i.LON Vision 2.0* read/write or application object to be added to your custom FPM configuration Web page.

**Note:** You cannot add Image Swapper or Check Box objects to your custom FPM configuration Web page.

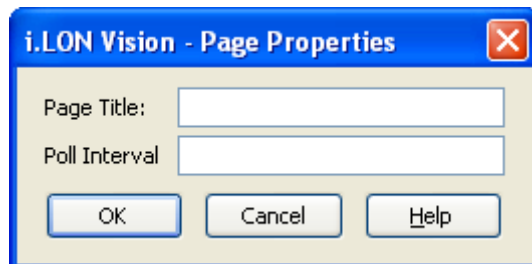
8. The respective dialog for the selected *i.LON Vision 2.0* object opens.



9. Specify the data point to be monitored and controlled by the *i.LON Vision 2.0* object. You can either click **Prog Name** and manually enter the name of the data point, or you can click **Select** and use the **Data Points** dialog to select the data point following these steps:
  - a. In the **Data Points** dialog, click **Prog Name**. This enables all instances of your FPM application to write to their respective data points. You must select this option or your FPM configuration Web page will not function properly for other instances of the FPM.
  - b. Expand the network, channel, device, and then the functional block of the data point to be monitored, and then click the desired data point. If the data point has a structured type, you can expand the data point and select a field within the structure.
  - c. Click **OK** to return to the object dialog. The **Name** property is updated with the programmatic name of the data point. The **Format** property is updated to show the format description of the selected data point. The format description consists of the data point's program ID; SNVT, SCPT, UNVT, UCPT, or built-in data type; and format (e.g., SI metric or US customary).




10. Configure the other object properties in the dialog following the *i.LON Vision 2.0 User's Guide*, and then click **OK**.
11. Repeat steps 7–10 to add other i.LON Vision 2.0 objects to your custom FPM configuration Web page.
12. Optionally, you change the title of your FPM configuration Web page. The default page title is **NLS\_TEXT**. To change the page title, follow these steps:
  - a. Click **Format** and then click **Page Title Properties**. The **Page Properties** dialog opens.



8. In the **Page Title** box, enter a descriptive page title and then click **OK**.

**Tip:** You can translate the page title into a number different languages using the *i.LON SmartServer 2.0 Programming Tool*. See *Creating Localized FPM Configuration Web Pages* in Chapter 8 for more information on how to do this.

13. Click **Publish** on the Editor toolbar () , click **File** and then click **Publish**, or click the **Browse** tab to publish your custom Web page to save the current draft of your custom FPM configuration Web page.
15. View your custom FPM configuration Web page from the SmartServer Web interface. To do this, click the **General** button above the navigation pane on the left side of the SmartServer Web interface, and then click the functional block representing your FPM application.

**Note:** If the FPM configuration Web page in the SmartServer Web interface does not display the draft you published with Contribute, you need to clear your browser's cache. To do this, follow these steps (for Internet Explorer 7):

- a. On the Internet Explorer **Tools** menu, click **Internet Options**. The **Internet Options** dialog opens to the **General** tab.
- a. Under **Browsing History**, click **Delete** to open the **Delete Browsing History** dialog.
9. In the **Temporary Internet Files** section, click the **Delete Files** button. Click **Yes** to confirm the deletion of the files. All the files that are currently stored in your cache are deleted.
10. Click **Close**, and then click **OK** to exit.
11. Press F5 or click the refresh button on the Internet Explorer toolbar to refresh the screen
16. You can now use your custom FPM configuration Web page to read and write values to the data points in your FPM application.

## Updating FPMs

The following section describes how to update your FPMs. The steps you perform depend on which component of your FPM you want to update: data point declarations (resource files), the FPM application (the source file), or the device interface.


### Updating Data Point Declarations

You can add new network variable and configuration property members to the UFPT used by your FPM, or update existing members and then add or update the data points in the source file (**.cpp** extension).

To update the data point declarations in your source file, follow these steps:

1. Use the NodeBuilder Resource Editor to generate an updated resource file set for your company. See Chapter 3 for more information on generating an updated resource file set.



2. Upload your company's updated resource file set to the root/LonWorks /types/User/<Your Company> folder on the SmartServer flash disk.
3. Use the *i.LON SmartServer 2.0 Programming Tool* to manually import the new or updated data point declarations. In the **LonMark Resource View**, right-click the UFPT from which the FPM project was created, and then click **Import All Declarations** on the shortcut menu. Alternatively, you can click the UFPT and then click the Import Declare All Data Points icon () at the top of the **LonMark Resource View**.
3. Continue to the next section, *Updating FPM Applications and Drivers*. Note that if you are updating an FPM application that uses a static interface, you must also update the device interface (XIF) file, as described in *Updating Device Interfaces*.

### *Updating FPM Applications and Drivers*

You can use the *i.LON SmartServer 2.0 Programming Tool* to modify the source file (.cpp extension) of your FPM application or driver. After you have finished modifying the code, you can upload the updated FPM to your SmartServer with the *i.LON SmartServer 2.0 Programming Tool*. To do this, follow these steps:

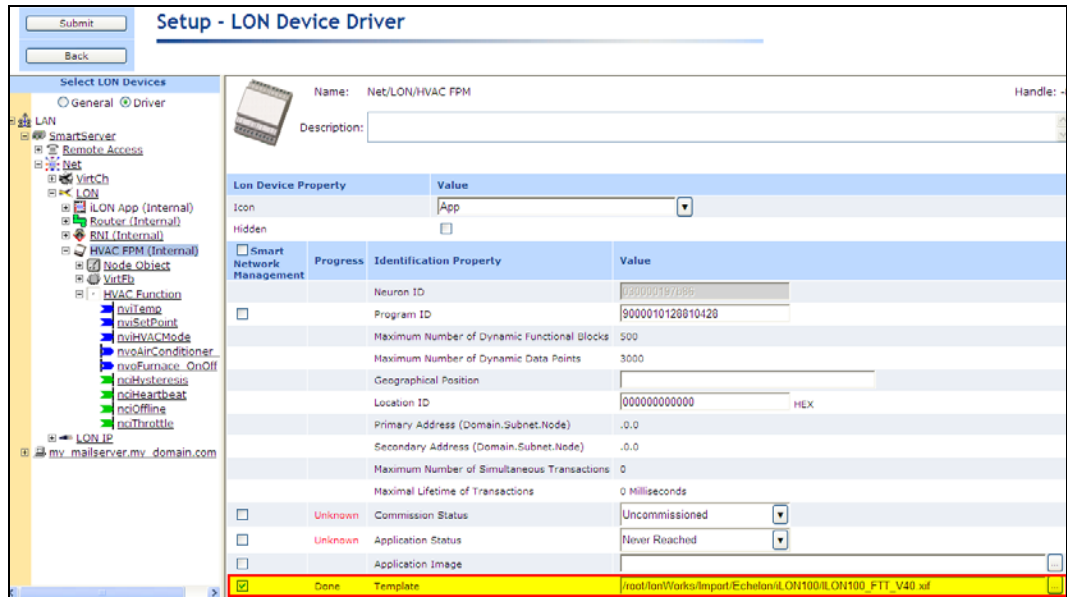
1. In the **C/C++ Projects** view of the *i.LON SmartServer 2.0 Programming Tool*, expand the **Release** folder, right-click the <company program ID>.UFPT<FPM name>.app || .drv file and then click **Transfer to i.LON SmartServer** in the shortcut menu.
2. The **Install FPM Module** dialog opens with the Deployment Settings window.
3. Optionally, you can modify the properties in the window as described in *Uploading FPM Applications and Drivers* earlier in this chapter.
4. Click **Finish** to upload your updated FPM to your SmartServer.
5. If you are deploying an updated FPM application, the current FPM executable module (.app extension) is stopped and unloaded, and the updated module is then loaded and initialized.
6. If you are deploying an updated FPM driver, reboot the SmartServer to initialize the updated module.

### *Updating Device Interfaces*

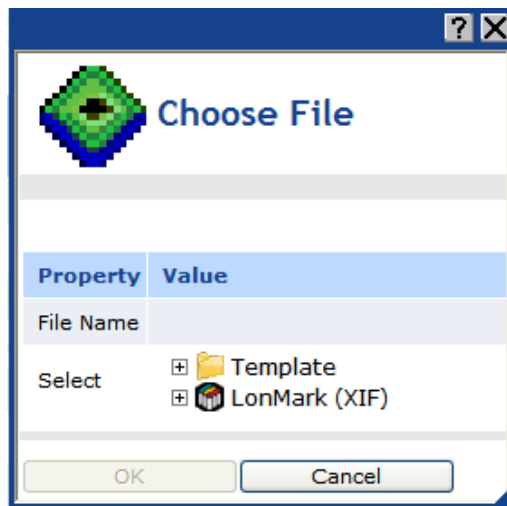
You can update the static device interface (XIF) file used by a FPM application, and you can change the device interface used by an FPM device from a dynamic interface to a static interface and vice versa.

To update the device interface and activate it in on the SmartServer, follow these steps:

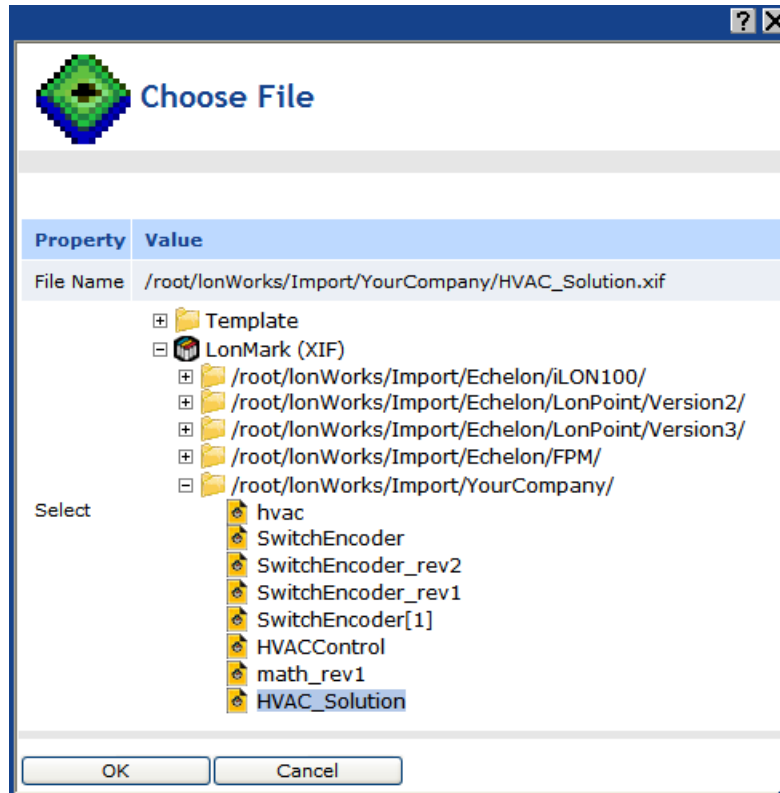
1. Create a new model file or update an existing model file, use *i.LON LonWorks Interface Developer* tool to convert the model file to a new XIF file, and then upload the new XIF file to the root/LonWorks/Import/<YourCompany> folder on the SmartServer flash disk. See Chapter 4 for more information on these steps.
2. Verify that you have selected a network management service mode as described in *Selecting a Network Management Service* earlier in this chapter.
3. Click **Driver**.
4. Select one or more devices from the tree to be upgraded.
  - To select one device, click that device. The **Setup - LON Device Driver** Web page opens.
  - To select multiple devices and perform a batch upgrade, click one device and then either hold down CTRL and click all other devices to be upgraded or hold down SHIFT and select another device to upgrade the entire range of devices. The **Setup - LON Device Driver** Web page opens.



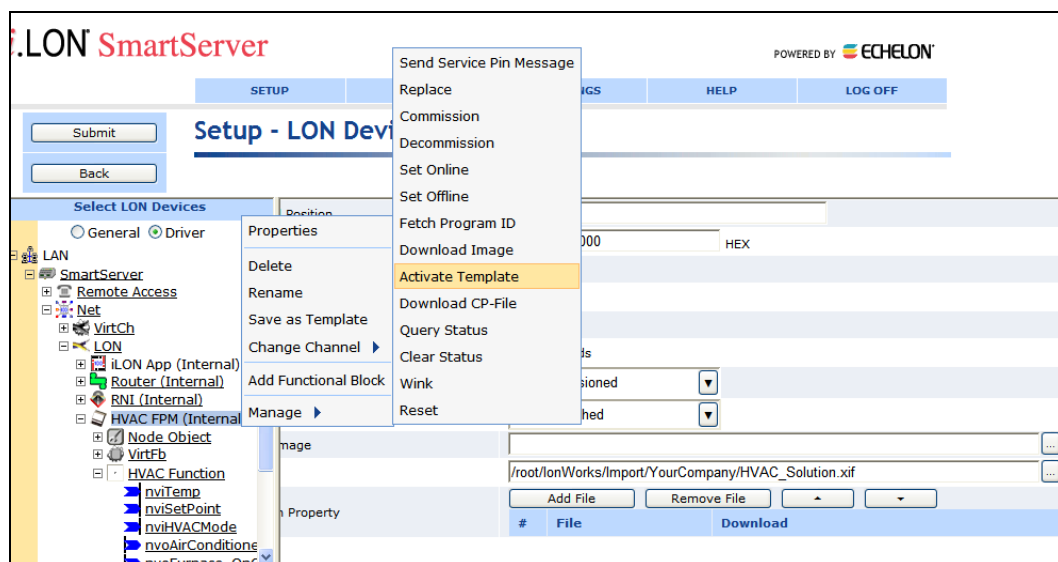
5. Select the XIF file to be activated, following these steps:
  - a. In the **Template** property, click the button to the right.
  - b. The **Choose File** dialog opens.



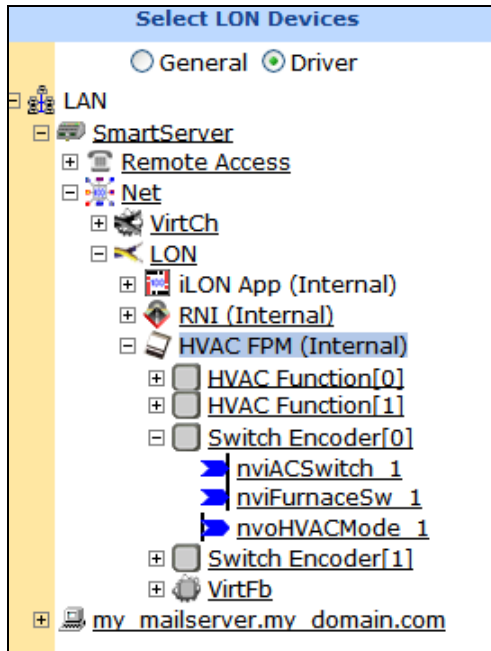
- c. Expand either the **LonMark (XIF)** folder, expand the subfolders containing the XIF file to be loaded onto the SmartServer, and then click the XIF file.



- d. Click **OK** to return to the **Setup - LON Device Driver** Web page.
  - e. Click **Submit**.
6. Right-click one of the selected devices in the SmartServer tree, point to **Manage**, and then click **Activate Template** in the shortcut menu. Alternatively, you can clear and then select the **Smart Network Management** check box to the left of the **Template** property in the **Setup -LON Device Driver** Web page and then click **Submit**.



7. You must wait approximately 15 seconds for the SmartServer to instantiate the updated XIF file. Once the XIF has been instantiated, you can expand the FPM device to see the functional blocks and data points in the updated XIF.



8. If you switched a static device interface to a dynamic interface, add a functional block representing your FPM application as described in the *Using a Dynamic Device Interface* section earlier in this chapter

---

## Deploying FPMs on Multiple SmartServers

After you have deployed FPMs on a development SmartServer, you can deploy the FPM applications and drivers you have developed on multiple SmartServers. To do this, each SmartServer on which an FPM application or driver is to be deployed, must have an FPM programming license from Echelon. You can then copy the files required to run the FPM applications and drivers to the SmartServer flash disk, reboot the SmartServer (FPM drivers only), and then create, commission, and connect the FPM devices on the SmartServers.

To deploy FPM applications on multiple SmartServers, you do the following:

1. Verify that an FPM programming license from Echelon is installed on each SmartServer.
2. Verify that you have the correct user name and password to access your SmartServer via FTP and that FTP access is enabled on your SmartServer. To do this, follow these steps:
  - a. Right-click the local **SmartServer** icon, point to **Setup**, and then click **Security** on the shortcut menu. Alternatively, you can click **Setup** and then click Security. The **Setup – Security** Web page opens.
  - b. In the **General** property, verify that the **FTP/Telnet User Name** and **FTP/Telnet Password** properties are correct.
  - c. In the **Service** property, verify that the **Enable FTP** check box is selected.
3. In the browser of an FTP client such as Microsoft Internet Explorer 7, enter the FTP URL of your SmartServer (ftp://192.168.1.222, for example).
4. Enter the FTP/Telnet user name and password for accessing your SmartServer via FTP.
5. Copy the following files to the listed folder on the SmartServer flash disk:
  - Copy your company’s resource file set for the FPM (.ENU, .fmt, .fpt, .ls, and .typ files) to the root/lonworks/types/user/<YourCompany> folder.

- If you are deploying an FPM application that uses static functional blocks, copy the device interface (XIF) file (.xif extension) to the root/lonworks/import/<YourCompany> folder.
  - Copy the FPM executable modules (.app or .drv extension) to the root/modules/user/<YourCompany> folder.
  - If you created a custom FPM configuration Web page for your FPM applications, copy your custom Web pages to the root/web/config/Fb folder.
6. If you are deploying an FPM driver, reboot the SmartServers.
  7. Create, commission, and connect the FPM devices on the SmartServers as described in this chapter.

---

## Deploying Licensed FPM Applications

You can deploy Echelon first-party FPM applications and third-party FPM applications on your SmartServer. To do this, your SmartServer must not only have an FPM programming license from Echelon, but it must also have a separate FPM application license from Echelon or the third-party FPM vendor for the FPM application being deployed on your SmartServer.

To deploy an Echelon first-party FPM application or a third-party FPM application on your SmartServer, you do the following:

1. Verify that the license for the Echelon first-party FPM or third-party FPM to be deployed is installed on the SmartServer. Echelon or the third-party FPM vendor should provide you with instructions on how to install their FPM application licenses on your SmartServer.
2. Verify that you have the correct user name and password to access your SmartServer via FTP and that FTP access is enabled on your SmartServer. To do this, follow these steps:
  - a. Right-click the local **SmartServer** icon, point to **Setup**, and then click **Security** on the shortcut menu. Alternatively, you can click **Setup** and then click Security. The **Setup – Security** Web page opens.
  - b. In the **General** property, verify that the **FTP/Telnet User Name** and **FTP/Telnet Password** properties are correct.
  - c. In the **Service** property, verify that the **Enable FTP** check box is selected.
3. In the browser of an FTP client such as Microsoft Internet Explorer 7, enter the FTP URL of your SmartServer (ftp://192.168.1.222, for example).
4. Enter the FTP/Telnet user name and password for accessing your SmartServer via FTP.
5. Copy the following files to the listed folder on the SmartServer flash disk:
  - Copy the resource file set (.ENU, .fmt, .fpt, .ls, and .typ files) provided by Echelon or the third-party FPM vendor to the root/lonworks/types/<YourCompany> folder.
  - If you are deploying an FPM application that has a static interface, copy the device interface (XIF) file (.xif extension) provided by Echelon or the third-party FPM vendor to the root/lonworks/import/<YourCompany> folder.
  - Copy the FPM executable module (.app extension) provided by Echelon or the third-party FPM vendor to the root/modules/User/<YourCompany> folder.
  - If a custom FPM configuration Web page was created for the FPM, copy the .htm files provided by Echelon or the third-party FPM vendor to the root/web/config/Fb folder.
6. If you are deploying an FPM driver, reboot the SmartServer.
7. Create, commission, test, and connect the FPM on your SmartServer as described in this chapter.



## Creating FPM Application Licenses

This chapter describes how to create licenses for your FPMs so that customers can order and implement your FPMs on their SmartServers. It describes how to build an FPM licensing tool. It explains how to enable a license validation feature in your FPM application. It describes how to create FPM licenses. It lists the files you need to provide to customers who order your licensed FPM applications.

---

## Licensing Overview

You can create FPM application licenses and let customers order and implement your FPMs on their SmartServers. To create an FPM application license and make your FPM application available for order, you do the following:

1. Create an FPM licensing tool.
2. Enable license validation in your FPM application.
3. Build the release version of your licensed FPM application.
4. Create FPM application licenses.
5. Supply your customers the FPM licenses, FPM applications, your FPM library, and your resource files.

---

### *Creating an FPM Licensing Tool*

The *i.LON SmartServer 2.0 Programming Tools* includes an *i.LON License Generator* program that you can use to construct your own FPM licensing tool. The *i.LON License Generator* is located in the `LonWorks\iLON\Development\Licensing\iLONLicenseGen` folder, and it includes the following three components:

- The main executable (**iLONLicenseGen.exe**) that provides a user interface for entering the values used to generate an FPM license.
- A sample license generator configuration file (an XML file named **iLONLicenseGenValuesSample.xml**) that demonstrates the structure of the *i.LON License Generator* user interface and provides sample pre-defined values.
- A sample security DLL file (**LicenseSecurityHMACMD5.dll**) that takes the values entered in the *i.LON License Generator* user interface and creates an FPM license.

To construct your FPM licensing tool, you create a license values file that defines the default values for the user interface of the *i.LON License Generator*, and you create a security DLL file named **LicenseSecurity.dll** that takes the values entered in the *i.LON License Generator* user interface and creates an FPM license. If you do not have the resources to build the security DLL file, you can rename the sample DLL file to **LicenseSecurity.dll**. This file provides a standard HMAC-MD5 digest security algorithm.

Once you create the license manager file (**iLONLicenseGenValues.xml**) and the security DLL file (**LicenseSecurity.dll**), and you have built a release version of your licensed FPM application, you can open the *i.LON License Generator* and begin using it to create FPM licenses.

#### *Creating a License Generator Configuration File*

The *i.LON License Generator* requires an XML configuration file named **iLONLicenseGenValues.xml** that defines the default values for the properties in the user interface. A sample configuration file named “**iLONLicenseGenValuesSample.xml**” is provided in the `LonWorks\iLON\Development\Licensing\iLONLicenseGen` folder. The sample file displays the XML structure of that the *i.LON License Generator* requires. In addition, the sample configuration file provides an example of the default values that you can define for your FPMs.

You can create your own default values using the sample configuration file as a guide, or you can copy and rename the sample file to **iLONLicenseGenValues.xml** and modify the values to fit your company and your FPMs.

The configuration file includes a `<PredefinedFeatures>` element in which you define a `<Company>` element. Within the `<Company>` element, you define (1) your company’s properties such as name, and LonMark ID, and (2) one or more features, the properties for each feature such as name, algorithm, and secret key, and the pre-defined values for the properties. The following table lists the company and feature properties you define within the `<PredefinedFeatures>` element. An example that



demonstrates the structure of the <PredefinedFeatures> element and the properties you can define in it is provided after the table.

Property	Description
<b>Company</b>	
CompanyName	Specify the name of your company.
ShortCompanyName	<p>Specify the name of your company or an abbreviated name. Legal characters are those that would be legal for file names on the SmartServer. You should only use letters, numbers or the underscore character. Do not include spaces in this name.</p> <p>The ShortCompanyName will appear at the beginning of the name of your FPM license file.</p>
LonMarkID	<p>Specify your company's LONMARK manufacturer identifier (MID). The LonMarkID will appear after the ShortCompanyName in the name of your FPM license file. If you do not have a LONMARK ID, you can use an arbitrary number.</p> <p>This field prevents license naming collisions from occurring because of a common company name. It is useful for scenarios in which a large company has multiple divisions that each have their own LONMARK ID.</p>
<b>Company/Feature</b>	
FeatureName	<p>Specify a descriptive name that uniquely identifies the FPM. For example, you can specify "FPM Math Functions" or "FPM HVAC Controller" for the math and HVAC examples used in this guide.</p> <p>The specified value will be listed in the <b>Feature Name</b> field of the FPM application license.</p> <p>The first feature you specify in the configuration file and the specified default values of its associated properties will appear in the <i>i.LON License Generator</i> by default.</p> <p>The values that appear in the <i>i.LON License Generator</i> are based on the selected FeatureName.</p>
ShortFeatureName	<p>Specify a condensed or abbreviated name for your FPM. The same constraints and recommendations about legal characters from the ShortCompanyName also apply to this field.</p> <p>For example, you can specify "Math" or "HVAC" for the math and HVAC examples used in this guide.</p> <p>The ShortFeatureName will appear after the LonMarkID in the name of your FPM license file.</p>
AlgorithmIndex	<p>Specify an index corresponding to an algorithm index in your license security source file. This index determines which security algorithm in the security DLL file is run. You will probably only have one algorithm, in which case an index of 0 or 1 would be appropriate.</p> <p>The specified AlgorithmIndex will appear in the <b>Secure Algorithm Index</b> field of the <i>i.LON License Generator</i> when its associated feature is selected in the <b>Feature Name</b> field.</p> <p>The sample security DLL file uses an algorithm that has an index of <b>0</b>. If you plan on using the sample security DLL file, you must specify <b>0</b> in the AlgorithmIndex property or else the <i>i.LON License Generator</i> will not be able to generate an FPM license. See <i>Building the Security DLL File</i> for more information on using algorithm</p>

	indexes.
LicenseType	Specify the type of license to be issued. The default LicenseType is “Unlimited”. You may create your own license type designations. For example, you could specify a “Demo” license if you plan on modifying your FPM application so that the license provided to a customer expires after a specified trial period such as 30 days.
LockType	Specify the type of lock used to uniquely identify a customer’s SmartServer. You can specify one of the following four lock types: <ul style="list-style-type: none"> <li>• <b>MACID.</b> The unique 12-digit hexadecimal Ethernet MAC address assigned to each SmartServer. Using the MACID ensures that the FPM license you are issuing is associated with a specific SmartServer. This is the recommended lock type.</li> <li>• <b>LUID.</b> Any one of the sixteen unique 12-digit hexadecimal Neuron IDs assigned to each SmartServer. Using the LUID ensures that the FPM license you are issuing is associated with a specific SmartServer. You can use this lock type instead of the MACID.</li> <li>• <b>User.</b> Some user-defined identifier.</li> <li>• <b>None.</b> No lock type is used.</li> </ul>
Options	Optionally, you can enter any text in this property. For example, you could specify the length of a trial period of an FPM, such as “D- 30” for 30 days.
SecretKey	Specify a unique hexadecimal string that functions as the secret key for the feature. The secret key is used by the security DLL file in calculating the unique license key for the FPM license. You may specify a different secret key for each feature. The length of the secret key must be appropriate for the security algorithm used.

```

<PredefinedFeatures>
  <Company>
    <CompanyName>Our Corporation</CompanyName>
    <ShortCompanyName>Our Company</ShortCompanyName>
    <LonMarkId>0</LonMarkId>
    <Feature>
      <FeatureName>FPM HVAC Controller</FeatureName>
      <ShortFeatureName>HVAC</ShortFeatureName>
      <AlgorithmIndex>0</AlgorithmIndex>
      <LicenseType>Unlimited</LicenseType>
      <LockType>MACID</LockType>
      <SecretKey>5BD6217EA180AA116A51AAD1D9A6DD1E</SecretKey>
    </Feature>
    <Feature>
      <FeatureName>FPM Math Function</FeatureName>
      <ShortFeatureName>Math</ShortFeatureName>
      <AlgorithmIndex>0</AlgorithmIndex>
      <LicenseType>Unlimited</LicenseType>
      <LockType>MACID</LockType>
      <SecretKey>22222222222222222222222222222222</SecretKey>
    </Feature>
  </Company>
</PredefinedFeatures>

```

## Creating a Security DLL File

The *i.LON* License Generator requires a security DLL file named **LicenseSecurity.dll** that implements a specific security algorithm. The security algorithm enables you to publish unique digital signatures for your FPM licenses. You can build your own **LicenseSecurity.dll** file, or you can use the sample security DLL file provided by Echelon. The following sections describe how to build the security DLL file and how to use the provided sample security DLL file.

### Building the Security DLL File

To build your own security DLL file, you can use the sample license security file provided with the *i.LON* SmartServer 2.0 Programming Tools (demo or full version). This C++ source file, **LicenseSecuritySample.cpp**, is located in the `LonWorks\iLON\Development\Licensing\iLONLicenseGen` folder. It documents the proper interface to be used to create the security DLL file.

In this sample, the `GenerateLicenseKey()` routine is called when the **Create License** button in the *i.LON* License Generator is clicked. This routine does the following:

1. Receives the following seven parameters: `algorithmIndex`, `pText`, `textLen`, `pSecretKey`, `keyLen`, `ppLicenseKey`, and `pLicenseKeyLen`.
  - `algorithmIndex` is the value from the **Algorithm Index** field of the *i.LON* License Generator.
  - `pText` is a pointer to a character array in which the text entered in the **Issuing Company Name**, **Feature Name**, **License Type**, **Lock Type**, and **Lock ID** fields of the *i.LON* License Generator is stored.
  - `textLen` is an integer that stores the length of the text referenced by `pText`.
  - `SecretKey` is a pointer to a byte array in which the binary value of the text entered in the **Secret Key** field of the *i.LON* License Generator is stored.
  - `keyLen` is an integer that stores the length of the byte array referenced by `SecretKey`.
  - `ppLicenseKey` is a pointer to a pointer to byte array where the returned license key will be stored.
  - `pLicenseKeyLen` is a pointer to an integer that will store the length of the returned license key.
2. Checks the algorithm index passed in from the *i.LON* License Generator. The algorithm index determines which security algorithm is run. You can have multiple algorithm indexes to handle different features or situations. This sample includes one security algorithm that is run when the algorithm index is 0.
3. Executes the specified security algorithm. This algorithm must use the `pText`, `textLen`, `pSecretKey`, and `keyLen` parameters and generate the license key.

**Note:** C source files for open source implementations of the MD5 and HMAC-MD5 digest algorithms are included in the `LonWorks\iLON\Development\Licensing` folder. You can use these source files to implement your security algorithm for the security DLL and the FPM. You can use other available security algorithms such as SHA-1, SHA-256, or DES in your security DLL and FPM.

**Echelon makes no recommendation about the suitability of any of these algorithms. The algorithms are provided for demonstration purposes only.**

If you use the provided source files, you may include them in your security DLL source file using the `#include` statement, or you may compile them separately. In either case, you should define

the C macro `DONT_TRANSLATE_NAMES` before each of these files. The names of the routines to be used all begin with “`LICMGR_`”

If you use the supplied implementation of the HMAC-MD5 digest algorithm, you can optionally turn it into a non-standard algorithm by adjusting the pre-defined `HMAC_IPAD_XOR_VALUE` and `HMAC_OPAD_XOR_VALUE` values. If you want to modify the pad XOR values, you must define them as one-byte hex values. Creating a non-standard algorithm may be useful if you want to further protect your FPM application in the event the secret key defined for your FPM is compromised. However, this theoretically may reduce the strength of the digest.

**Echelon makes no recommendation regarding the use of this feature.**

4. Returns the license key to the *i.LON* License Generator, which converts the license key to ASCII text and then generates the .XML file to be supplied to your customers. You must set the expression `*ppLicenseKey` to point to the memory area containing the binary license key (the “digest”) generated by your security algorithm, and you must set the expression `*pLicenseKeyLen` to the byte length of the binary license key.

### Using the Sample Security DLL File

If you do not have the resources to build the security DLL file, you can use the sample pre-built security DLL file provided with the *i.LON* SmartServer 2.0 Programming Tools (demo or full version). This executable file, **LicenseSecurityHMAMD5.dll**, is located in the `LonWorks\iLON\Development\Licensing\iLONLicenseGen` folder.

To use this file, you just rename it to “**LicenseSecurity.dll**”. This file implements a standard HMAC-MD5 digest algorithm from “open source” code. The source code for this sample DLL is in the file **LicenseSecurityHMAMD5.cpp**, located in the `LonWorks\iLON\Development\Licensing\iLONLicenseGen` folder. Note that this sample security algorithm is not guaranteed to be suitable for any particular scenario.

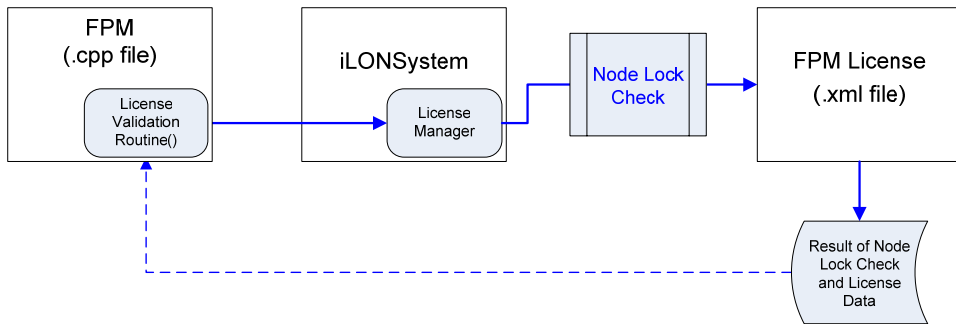
**The user is solely responsible for selecting a security algorithm in the security DLL file and in the FPM application. Echelon makes no recommendations regarding the suitability of any security algorithm.**

---

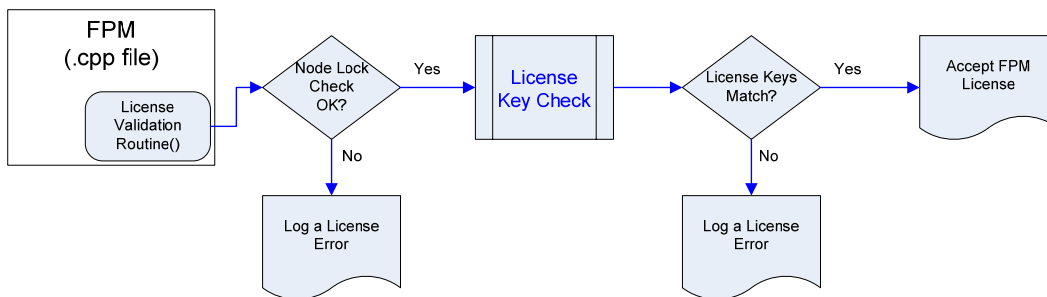
## *Enabling License Validation in an FPM Application*

You need to modify your FPM application so that it can check whether a customer’s SmartServer has a valid license for running your FPM. This entails writing a separate license validation routine in the source file of your FPM (.**cpp** extension). Your license validation routine must perform two major tasks: (1) verify that the Lock ID (MACID, LUID, or other user-defined lock type) specified in the FPM license file matches the one on the customer’s SmartServer, and (2) verify that the license key in the FPM license file is valid. The FPM license is the .xml file you created with the *i.LON* License Generator as described in *Creating FPM Licenses* in this chapter. This file must be installed in (or under) the `root/config/license` folder on the SmartServer flash disk.

To verify the lock ID, your license validation routine must call the built-in license manager on the SmartServer and provide it the license ID (Company Name and Feature Name) specified in the FPM license file, and the file path of the FPM license relative to the `root/config/license` folder on the SmartServer flash disk. The license manager will then check that the Lock ID in the FPM license matches the one on the SmartServer (this is referred to as the Node Lock Check). The license manager then returns the result of the Node Lock Check and the data in the FPM license file to your license validation routine. The license data is returned as a structure that contains the license type, lock type, lock ID, options, license key, and the length of the license key.



To verify that the license key in the FPM license file is valid, your license validation routine needs to first check the results of the Node Lock Check. If the SmartServer passed the Node Lock Check, your license validation routine should then check the FPM license data; otherwise, it should log a license error. To check the FPM license data, your license validation routine must call the same security algorithm you used to create the FPM license, and it must provide the security algorithm (1) the license data returned by the license manager and (2) the secret key assigned to the FPM in the license configuration file (**iLONLicenseGenValues.xml**). The security algorithm then calculates a license key. The license validation routine should then compare the license key stored in the FPM license file to the one just calculated. If their sizes and values match, the license validation routine should accept the FPM license; otherwise, it should log a license error.



Overall, to enable license validation in your FPM application, you perform the following steps:

1. Insert `#include` directives for the required file **LicenseMgr.h** and optionally the security algorithm files **LicMgrMd5.c** and **LicMgrHMacMd5.c**. In addition, you must redefine the name translations the macros defined at the beginning of the **LicMgrMd5.c** and **LicMgrHMacMd5.c** files.
2. Declare data variables for the FPM license status and for the secret key assigned to the FPM.
3. Create a license validation routine.
4. Write the license validation algorithm to do the following:
  - a. Define local variables for the data to be passed to and returned by the license manager and by the security algorithm, declare that the FPM is not licensed, and declare a license structure for the license data to be passed to and returned by the license manager.
  - b. Call the method in the *i.LON* License Manager that performs a Node Lock Check and returns the node lock status and license data.
  - c. Check the results of the Node Lock Check. If the SmartServer passes the check, call a security algorithm that calculates the license key based on the license data. Compare the license key stored in the license file to license key returned by the security algorithm. Based on the results of the license key evaluation, validate the FPM license, or log a license error.
5. Implement some mechanism in your FPM application that results in the license validation routine being called. You are solely responsible for implementing this mechanism.
6. Compile your licensed FPM application.

### Tips for Securing your Licensing Scheme:

You can implement a number of security measures in your licensing scheme to help protect your FPM application from unauthorized use or piracy. The examples provided in this section demonstrate some of these measures, which are designed to force anyone attempting to break the licensing scheme to disassemble your object code and reverse-engineer the algorithms. To test the strength of your licensing scheme, you should attempt to break it once you have completed it. This will help you identify any weaknesses in your licensing scheme.

Note that there are instances in the examples where a number of techniques can be used in implementing a security measure, but no specific recommendation is made as to which technique is the best. This is because it is not clear which technique provides the best security. Furthermore, if every customer follows the same technique, the overall security of the FPM applications being manufactured is weakened.

Once you enable license validation in your FPM application and build your FPM, it is recommended that you remove the internal-only symbols from your FPM application. This obscures the location of the routines and data in your FPM application. The examples in this section assume that you will do this; therefore, the provided security functions and data are declared as “static”. Note that if you intend to remove the internal-only symbols, some of the techniques described in this section are optional (e.g., translating names via macros); however, it is recommended that you still implement them. This is because the techniques provide a level of security in the event the removal of the internal-only symbols is not performed. For instructions on how to remove the internal-only symbols from your FPM application, see *Building the Release Version of a Licensed FPM Application* later in this chapter.

Note that no software protection scheme is completely secure. Individuals with the resources to break your licensing scheme may be able to eventually do so.

#### *Step 1: Inserting Include Directives and Macro Definitions*

The first step in enabling license validation in your FPM application is to insert `#include` directives for the files that contain the methods to be called by your FPM application. The only file that you must include is **LicMgr.h**. Optionally, you will need to include **LicMgrMd5.c** if you are using the supplied MD5 security algorithm in your FPM license validation routine, and both **LicMgrMd5.c** and **LicMgrHMacMd5.c** (in that order) if you are using the supplied HMAC-MD5 security algorithm.

Note that if you include the **LicMgrMd5.c** and **LicMgrHMacMd5.c** files, you must provide new definitions for the name translation macros defined at the beginning of those files. These macros begin with “LICMGR\_” and by default are defined as “CHANGE\_ME!” (an intentionally illegal value). You can redefine the macros directly in the files, or simply copy the `#define` statement for each one and put the new definitions in the source file (`.cpp` extension) or header file (`.h` extension) of your FPM application. The new definitions must be inserted before the included source files.

The **LicMgrMd5.c** file contains six macros for which you need to supply definitions, and the **LicMgrHMacMd5.c** file contains one macro. These macros redefine the names of the actual security algorithm routines to obscure them from malicious attempts to bypass your licensing. To maximize the protection provided by this mechanism, the name definitions should look related to your FPM application, but they should obscure their actual functions.

It is also recommended that you obscure your own symbol names. All the symbols in the provided examples that use all upper-case are intended to be defined as macros, with the true name being intentionally misleading.

To insert the `#include` directives in your FPM application, do the following.

1. Start the *i.LON SmartServer 2.0 Programming Tool*. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0 Programming Tool* opens.
2. In the **C/C++ Projects** view, expand the FPM project folder if the header file (`.h` extension) for your FPM application is hidden.

3. Click the header file. The header file view opens to the right of the **C/C++ Projects** view.
4. Insert the following include directives in the “includes go here.” section of the header file.

```
// Required include statement. Finds FPM license and performs
// node lock check.
#include "LicenseMgr.h"

// Define name translations for macros in MD5 digest if not
// already defined in file.
#define LICMGR_MD5Init <name>
#define LICMGR_MD5Update <name>
#define LICMGR_MD5Final <name>
#define LICMGR_MD5Transform <name>
#define LICMGR_MD5Encode <name>
#define LICMGR_MD5Decode <name>

// Optional include statement(but required if using MD5 or
// HMAC-MD5). Uses MD5 digest to generate a license key.
#include "LicMgrMd5.c"

// define name translation for macro in HMAC-MD5 digest if not
// defined in file.
#define LICMGR_hmac_md5 <name>

// Optional include statement(required if using HMAC-MD5).
// Uses HMAC-MD5 digest to generate a license key
#include "LicMgrHmacMd5.c"
```

```
UFPETHVACController.cpp  UFPETHVACController.h
// -----
// includes go here.
// -----
#include <string>
#include "Standard_NVT.h"
#include "FPM_Variable.h"
#include "FPM_Starter.h"

#include "LicenseMgr.h"

#define LICMGR_MD5Init systGetx_a
#define LICMGR_MD5Update systGetx_b
#define LICMGR_MD5Final systGetx_c
#define LICMGR_MD5Transform systGetx_d
#define LICMGR_MD5Encode systGetx_e
#define LICMGR_MD5Decode systGetx_f
#include "LicMgrMd5.c"

#define LICMGR_hmac_md5 systGetx_g
#include "LicMgrHmacMd5.c"
```

### Step 2: Declaring Data Variables

After you insert the include directives, you need to declare data variables in the routine that are used to store the results of the Node Lock Check and the secret key assigned to the FPM. To declare the data variables, follow these steps:

1. Under the FPM project folder in the **C/C++ Projects** view, click the source file (**.cpp** extension) for your FPM application. The source file view opens to the right of the **C/C++ Projects** view.

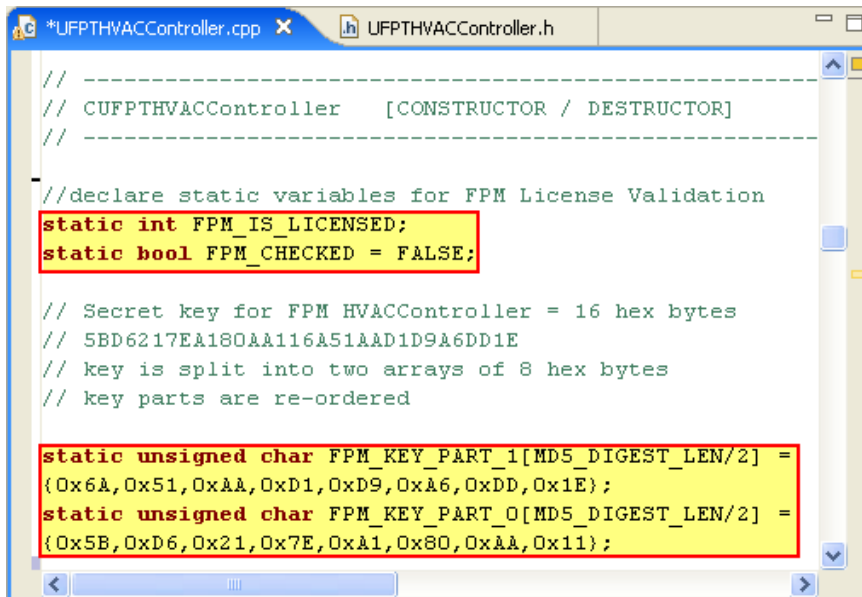
2. Insert code similar to the following somewhere after the FPM constructor:

```
// flag defining whether FPM license has been validated
static int FPM_IS_LICENSED; // initialized at run-time

// flag indicating whether FPM license has been checked
static bool FPM_CHECKED = FALSE;

// secret key for MD5/HMAC-MD5 algorithms
static unsigned char FPM_KEY_PART_1[MD5_DIGEST_LEN] =
{0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>,
0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>, 0x<hex>};
```

**Tip:** You could alternatively split the secret key into several pieces and put them back together in your validation algorithm. You could also put the key definition (or some of its pieces) inside your validation routine as a local variable. If a status variable is only accessed in a single routine but it needs to retain its value outside the routine, it may be placed inside the routine as a static local variable.



```
*UFPTHVACController.cpp x UFPTHVACController.h
// -----
// CUPTHVACController [CONSTRUCTOR / DESTRUCTOR]
// -----

//declare static variables for FPM License Validation
static int FPM_IS_LICENSED;
static bool FPM_CHECKED = FALSE;

// Secret key for FPM HVACController = 16 hex bytes
// 5BD6217EA180AA116A51AAD1D9A6DD1E
// key is split into two arrays of 8 hex bytes
// key parts are re-ordered

static unsigned char FPM_KEY_PART_1[MD5_DIGEST_LEN/2] =
{0x6A,0x51,0xAA,0xD1,0xD9,0xA6,0xDD,0x1E};
static unsigned char FPM_KEY_PART_0[MD5_DIGEST_LEN/2] =
{0x5B,0xD6,0x21,0x7E,0xA1,0x80,0xAA,0x11};
```

### Step 3: Creating the License Validation Routine

After you declare the static variables, you need to create a license validation routine. You first need to add your routine in the header file (.h extension) of your application. You can then place your license validation routine anywhere in your FPM application after the static variables you declared in step 2.

To create your license validation routine, you do the following:

1. Click the tab for the header file view.
2. Locate the Implements the user functionality section in the header file, and then insert the following code under the **public:** declarations.

```
void <FPM license validation routine name>();
```



```

// -----
//   Implements the user functionality
// -----
public:

// License Validation
void FPM_CHECK();

// Initialization and Cleanup
void Initialize();
void Shutdown();

// User implemented methods
void Work();
void OnTimer();
//void OnMyTimer();

```

**Tips:**

- You should obscure the name of the license validation routine (and other symbols shown in all capitals) to help secure your FPM application. To do this, define a macro with the same name as the symbol and then specify a name translation for it. For example, the following macro could be defined for the license validation routine created in step 2.

```
#define FPM_CHECK anotherFPMfunction
```

- You can alternatively create your license validation routine as a static file-scope routine. This enables the symbol name to be removed from the final object code module. Symbols for C++ class methods are always defined as global—even if they are a private class method—so that they cannot be as easily removed from the object code module. However, if you have a lot of class methods, using an intentionally misnamed method placed among the other methods may be sufficient.

3. Click the tab for the source file view. Insert the following code before or after the Initialize() routine:

```
void CUFPPT<FPM Name>::<FPM license validation routine name>()
{
}
```

```

// -----
// FPM License Validation Routine()

void CUFPPTHVACController::FPM_CHECK()
{
}

// -----
// CUFPPTHVACController::Initialize()
// Add anything that is necessary in order to get the module p
//
// This routine is called once, while starting up the module
void CUFPPTHVACController::Initialize()
{
// examples for setting up default/user-defined timer callback
hTimer1 = CreateTimer( FPM_TF_ONETIME, 10000); // calls th
// hTimer2 = CREATE_TIMER( FPM_TF_REPEAT, 3000, CUFPPTHVACCont
}

```

## Step 4: Writing the License Validation Algorithm

You need to write a license validation algorithm that (1) verifies that the Lock ID (MACID, LUID, or other user-defined lock type) specified in the FPM license matches the one on the customer's SmartServer, and (2) verifies that the license key in the FPM license file is valid.

To do this, your license validation algorithm must call the method in the *i.LON* License Manager that finds and parses an FPM license and performs a Node Lock Check. Your license validation algorithm then must evaluate whether the SmartServer passed the Node Lock Check. If the SmartServer passed the Node Lock Check, your method needs to calculate a license key and check whether the license key stored in the FPM license file matches the calculated license key. Your method must calculate a license key by calling the exact same security algorithm used in the License Generator security DLL and providing it the license data returned by the *i.LON* License Manager and the secret key defined for your FPM. The secret key for your FPM is stored in the variables that you declared in step 2.

You probably will only want to run the full license validation process once, so before proceeding you should check a flag (e.g., `FPM_CHECKED`), and you should set the flag somewhere in your license validation algorithm to indicate that the license has been checked.

Your license validation algorithm must declare local variables for the data to be passed to and returned by the license manager and by the security algorithm. Your license validation algorithm must specify the `CompanyName` and `FeatureName` properties defined in your FPM license file, and it must specify the file path of your FPM license file.

### Verifying the Lock ID

To write your license validation algorithm so that it verifies the Lock ID of a SmartServer, you do the following:

1. Check to see if you have already run the license check. For example:

```
if (!FPM_CHECKED)
{
    ...
}
```

2. Set a flag to indicate that your license validation algorithm has been called. This code does not need to be at the beginning. For example:

```
FPM_CHECKED = TRUE;
```

3. Declare the local variables for the data to be passed to and returned by the license manager and by the security algorithm. For example:

```
//data passed to and returned by license manager
LicMgrTaskCallBlock taskCallBlock;
LicMgrLicenseId licenseId;
LicMgrLicenseData *pLic;

//data passed to and returned by security algorithm
unsigned char secretKey[MD5_DIGEST_LEN];
unsigned char digest[MD5_DIGEST_LEN];
```

4. Enable the License Manager on the SmartServer to check for a valid FPM license. To do this, you store the `CompanyName` and `FeatureName` properties defined in your FPM license file in the `CompanyName` and `FeatureName` fields of the `LicMgrLicenseId` object. For example:

```
licenseId.szCompanyName = "Your Company Name";
licenseId.szFeatureName = "Your Feature Name";
```

Using the HVAC Controller FPM for example, the Company Name property would be “Our Corporation”, and the Feature Name would be “FPM HVAC Controller”. See *Creating a License*

*Generator Configuration File* for more information on the `CompanyName` and `FeatureName` properties.

5. Declare a License Manager control structure, then set the license file path and the license ID fields. The file path field corresponds to the path of the FPM license file relative to the `root/config/license` folder on the SmartServer flash disk, and the name of the FPM license file. The default file name of an FPM license is `<ShortCompanyName><LonMarkID><ShortFeatureName>.xml`. For example:

```
//declare license manager control structure
memset(&taskCallBlock, 0, sizeof(taskCallBlock));

//specify the license file path and license ID
taskCallBlock.pFilePath = "YourFilePath.xml";
taskCallBlock.pLicenseId = &licenseId;
```

Using the FPM license file of the HVAC Controller, for example, the file path property would be “OurCompany0HVAC.xml”. If you chose to place your licenses in a subfolder of the `/root/config/license` folder, the path name must contain that subfolder name, too (but not `/root/config/license`). For example, “OurCompanyFolder/OurFileName.xml”. See *Creating a License Generator Configuration File* for more information on the `ShortCompanyName`, `LonMark ID`, and `ShortFeatureName` properties.

6. Call the method in the *i*.LON license manger that finds the license and performs a node lock check. For example:

```
LICMGR_TaskCall_FindLicense(&taskCallBlock);
```

The following code demonstrates the lock ID of a SmartServer being checked by the license validation algorithm:

```
// FPM License Validation Routine()

void CUFPTHVACController::FPM_CHECK()
{
    // Set FPM_CHECKED flag to TRUE so method is called once
    FPM_CHECKED = TRUE;

    // declare local variables to be passed to and returned
    // by License Manager and Security Algorithm

    LicMgrTaskCallBlock taskCallBlock;
    LicMgrLicenseId licenseId;
    LicMgrLicenseData *pLic;
    unsigned char secretKey[MD5_DIGEST_LEN];
    unsigned char digest[MD5_DIGEST_LEN];

    // Check for a valid license.
    licenseId.szCompanyName = "Echelon Corporation";
    licenseId.szFeatureName = "FPM HVAC Controller";

    // Use dedicated license file
    memset(&taskCallBlock, 0, sizeof(taskCallBlock));
    taskCallBlock.pFilePath = "Echelon1HVAC.xml";
    taskCallBlock.pLicenseId = &licenseId;

    // Macro for indirect task call.
    LICMGR_TaskCall_FindLicense(&taskCallBlock);
```

## Verifying the License Key

To write your license validation algorithm so that it verifies the license key in the FPM application license file, you do the following:

1. Check the results of the Node Lock Check. For example:

```
if (taskCallBlock.sts == LicMgrStsOK)
{
    //store the license data
    pLic = taskCallBlock.pLicense;
    ...
}
```

2. If the SmartServer passes the check, get the secret key defined for your FPM in a block of memory. If you split it into pieces, you must assemble them here. For example:

```
memcpy(secretKey, FPM_KEY_PART_0, sizeof(FPM_KEY_PART_0));
memcpy(&secretKey[sizeof(FPM_KEY_PART_0)], FPM_KEY_PART_1,
sizeof(FPM_KEY_PART_1));
```

In this example the secret key is the 16-byte hexadecimal string appropriate for the HMAC-MD5 algorithm that you defined in step 2. See *Creating a License Generator Configuration File* for more information on the SecretKey property.

3. Call your security algorithm to calculate a license key. This must be the same security algorithm used in the License Generator DLL. The security algorithm returns a license key (digest) that is calculated from the license manager control structure data returned by the license manager, the length of the license manager control structure data, the secret key defined for your FPM, and the length of the secret key. For example:

```
LICMGR_hmac_md5((unsigned char*)pLic->szHashText,
strlen(pLic->szHashText), secretKey, 16, digest);
```

4. Compare the length and content of the license key stored in the license file to that of the license key returned by the security algorithm. For example:

```
if ((pLic->licenseKeyLen == MD5_DIGEST_LEN) &&
(memcmp(pLic->licenseKey, digest, MD5_DIGEST_LEN) == 0)){
```

5. Optionally, you can evaluate any other data in the license manager control structure data besides the licenseKey and licenseKeyLen fields. The license manager control structure contains all the data included in the license file, split into the following fields:
  - szHashText. A pointer to the text over which the security algorithm runs. It combines a number of other fields.
  - szLicenseType. A pointer to the text for the lock type (e.g., “MACID”).
  - lockType. An **enum** value indicating the lock type.
  - szLockId. A pointer to the text of the lock ID.
  - lockId. A pointer to the converted binary bytes of the lock ID if the lock type is “MACID” or “LUID”. If the lock type is not “MACID” or “LUID” or if this field is not used, this field is NULL.
  - szOptions. A pointer to the text of the options field, if any.
  - licenseKey. A pointer to the binary bytes of the license key. Used in step 4.
  - licenseKeyLen. the length in bytes of the licenseKey field. Used in step 4.
  - szFullLicenseText. A pointer to the complete XML text of the license.
  - szUserLicenseText. A pointer to the beginning of the (optional) user section of the XML license text.

**Note:** Pointers to strings will not be NULL if they are not used, instead some may point to empty strings.

6. Based on the results of the license key evaluation, validate the FPM license, or log a license error. You can use a value other than 1 and later check for that specific value instead of using a boolean zero/non-zero check. For example:

```
FPM_IS_LICENSED = SOME_MAGIC_NUMBER;
}
else
{
    taskCallBlock.pGeneric =
        (void*)"FPM HVAC license key is invalid\n";
    LICMGR_TaskCall_LogLicenseError(&taskCallBlock);
}
```

7. If the SmartServer did not pass the Node Lock Check, log a license error. For example:

```
}else
{
    taskCallBlock.pGeneric =
        (void*)"FPM HVAC license invalid or not
        found\n";
    LICMGR_TaskCall_LogLicenseError(&taskCallBlock);
}
```

8. Free any license data stored in memory. For example:

```
LICMGR_TaskCall_FreeLicenseData(&taskCallBlock);
}
```

The following code demonstrates the license key of an FPM application license file being checked by the license validation algorithm:

```
//Check if i.LON passed Node Lock Check
if (taskCallBlock.sts == LicMgrStsOK)
{
    //store the license data
    pLic = taskCallBlock.pLicense;

    // Copy the key in parts
    memcpy(secretKey, FPM_KEY_PART_0, sizeof(FPM_KEY_PART_0));
    memcpy(&secretKey[sizeof(FPM_KEY_PART_0)], FPM_KEY_PART_1,
    sizeof(FPM_KEY_PART_1));

    //call security algorithm
    LICMGR_hmac_md5((unsigned char*)pLic->szHashText,
    strlen(pLic->szHashText),
    secretKey, 16, digest);

    //compare license keys
    if ((pLic->licenseKeyLen == MD5_DIGEST_LEN) &&
    (memcmp(pLic->licenseKey, digest, MD5_DIGEST_LEN) == 0))
    {
        FPM_IS_LICENSED = SOME_NON_ZERO_VALUE;
        printf("***FPM license validated***\n");
    }
    else
    {
```

```

        char msg[100];
        sprintf(msg, "FPM license key is invalid: file
\\\"%s\\\"\\n\", taskCallBlock.pFilePath);
        taskCallBlock.pGeneric = (void*)msg;
        LICMGR_TaskCall_LogLicenseError(&taskCallBlock);
    }
} else
{
    char msg[100];
    sprintf(msg, "FPM license not found: file \\\"%s\\\"\\n\",
taskCallBlock.pFilePath);
    taskCallBlock.pGeneric = (void*)msg;
    LICMGR_TaskCall_LogLicenseError(&taskCallBlock);
}
// free the license data, if any
LICMGR_TaskCall_FreeLicenseData(&taskCallBlock);
}
}

```

### *Step 5: Implementing the License Validation Call Mechanism*

After you have written your license validation algorithm, you need to implement some mechanism in your FPM application that results in the license validation routine being called. This can be done in a variety of ways using the pre-defined code and routines in your FPM application; however, you are solely responsible for designing and implementing this mechanism.

### *Step 6: Compiling the Licensed FPM Application*

Once you have created the license validation call mechanism, you can compile your licensed FPM application. To compile your FPM, click **File** and then click **Save**. If your code has any errors, they will be listed with any warnings in the **Problems** view at the bottom of the document window. You can click on the errors and warnings listed in this view to debug your FPM. If the build is not performed, click **Project** and then click **Build Project**. You can then click **Project** and select **Build Automatically** so that your FPM applications are built automatically when you save them.

---

## ***Building the Release Version of a Licensed FPM Application***

After you enable license validation in your FPM application and compile it, you can build the release version of your FPM application that you will make available to customers. Building the release version entails removing the internal-only symbols from your FPM executable module (**.app** extension). Stripping the internal-only symbols greatly enhances the security of your licensing scheme, as it obscures the location of the routines and data in your code.

To remove the internal-only symbols from your FPM executable module, follow these steps:

1. Open a Command Prompt window.
2. At the command prompt, change the directory to the path of your FPM executable module.

For example, you type the following (without the break):

```

cd C:\LonWorks\iLON\Development\eclipse\workspace.fpm\
9000010000000000[3].UFPTHVACController/Release

```

3. Use the `set path` command to set the path to the directory containing the GNU strip utility (**strip.exe**). This utility is located in the  
C:\LonWorks\iLON\Development\eclipse\plugins\com.echelon  
.eclipse.ilon100.fpm\_0.9.0\compiler\3.3.2-vxworks-6.2\x86-win32\i586-wrs-vxworks\bin  
directory.

For example, you type the following (without the breaks):

```
set path=%path%;C:\LonWorks\iLON\Development\eclipse\plugins\
com.echelon.eclipse.ilon100.fpm_0.9.0\compiler\3.3.2-vxworks-
6.2\x86-win32\i586-wrs-vxworks\bin
```

4. Use the GNU `strip` command to remove the internal-only symbols from your FPM executable module. To do this, type the following command:

```
strip --strip-unneeded --target=elf32-big modulename
```

where *modulename* is the name of your FPM executable module in the following format:  
<company program ID>.UFPT<FPT Name>.app.

For example, you type the following (without the break):

```
strip --strip-unneeded --target=elf32-big
#9000010000000000[3].UFPTHVACController.app
```

5. Optionally, you can use the GNU `nm` command verify that the internal-only symbols have been removed from your FPM executable module. To do this, type the following command:

```
nm --numeric-sort modulename > someFileName.txt
```

For example, you type the following (without the break):

```
nm --numeric-sort #9000010000000000[3].UFPTHVACController.app >
noSymbols.txt
```

6. Use a text editor to open the file you created in step 5. You should observe that there are no internal-only symbols in the file. Internal-only symbols are denoted by a lower-case single letter in the second field (e.g., “t”, or “d”).

---

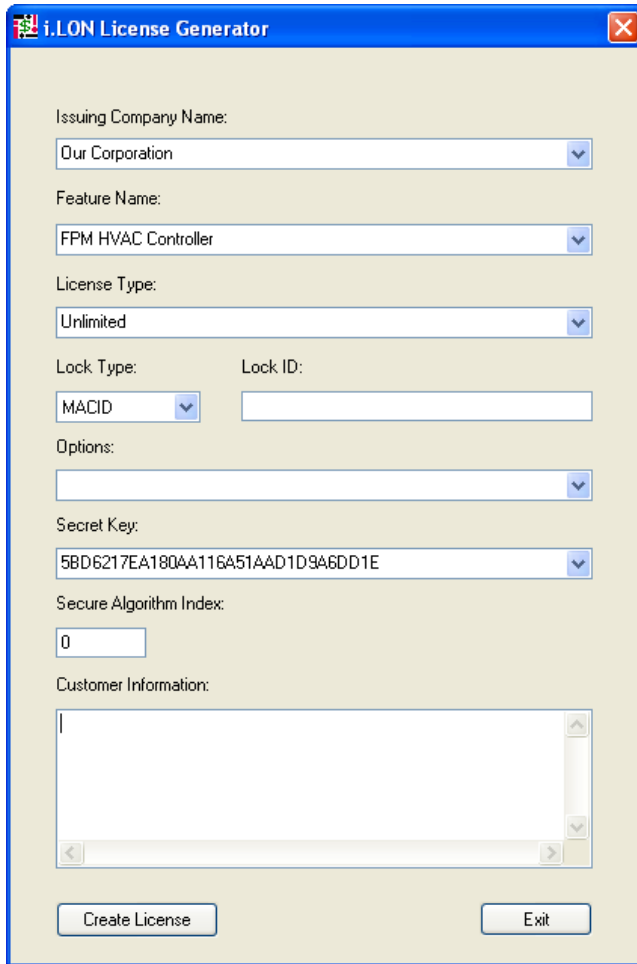
## Creating FPM Application Licenses

Once you create a license manger file (**iLONLicenseGenValues.xml**) and a security DLL File (**LicenseSecurity.dll**), and place these files in the same folder as the License Generator executable (**iLONLicenseGen.exe**), you can open the *i.LON* License Generator and begin using it to create FPM application licenses.

When you open the *i.LON* License Generator, you will observe that the first pre-defined feature specified in the configuration file and its associated default values appear in the dialog. You can select other pre-defined features from the **Feature Name** list and their specified default values will appear in the dialog.

To create FPM licenses using the *i.LON* License Generator, follow these steps:

1. Open the *i.LON* License Generator. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then select the **License Generator** folder. The `LonWorks\iLON\Development\Licensing\iLONLicenseGen` folder opens. Double-click the **iLONLicenseGen.exe** file. The **i.LON License Generator** opens.



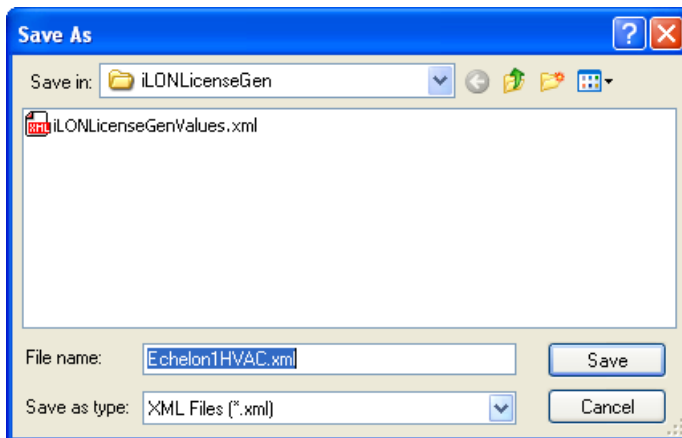
2. By default, the first pre-defined feature specified in the license configuration file (**iLONLicenseGenValues.xml**) and its associated default values appear in the dialog.
3. In the **Feature Name** property, select the FPM to be licensed from the list of the FPMs defined in the <PredefinedFeatures/Company/Feature/FeatureName> tags in the license configuration file. The specified default values of the properties associated with the selected FPM appear in the dialog.
4. In the **License Type** property, select the type of FPM license being issued. Typically, you will select **Unlimited**, but you can select **Demo** if you defined such property in the license configuration file and you plan on modifying your FPM application so that the license provided to a customer expires after a specified trial period such as 30 days.
5. In the **Lock Type** property, select the type of unique SmartServer identifier you requested from the customer. You should request the **MACID** (Ethernet MAC Address), but you can request the LUID (Neuron ID) or some other user-defined SmartServer identifier if you defined such property in the license configuration file.

The MACID and LUID are accessible from the SmartServer Web pages. You can help customers locate these identifiers using the SmartServer Web pages by providing the following instructions:

- To locate the MACID (Ethernet MAC Address) from the SmartServer Web pages, right-click the local SmartServer, point to **Setup**, and then click **System Info** (alternatively, you can click **Setup** and then click **System Info**). The **Setup – System Info** Web page opens. The **Ethernet MAC Address** is the first property listed under the **Ethernet** header.



- To locate the LUID (Neuron ID) from the SmartServer Web pages, click **Driver** at the top of the tree in the sidebar (left) frame, expand the **Net** network, expand the **LON** channel, and then click any internal SmartServer device, which have “(Internal)” appended to their names. The **Setup – LON Device Driver** Web page opens. The **Neuron ID** is the first property listed under the **Identification Property** header.
6. In the **Lock ID** property, enter the unique SmartServer identifier provided by your customer (MACID, LUID, or other user-defined identifier). If you are entering a MAC ID or LUID, you can enter the 12-digit hexadecimal number as a single string, or you can separate the hex digit pairs with dashes, spaces, colons, semi-colons, periods. For example, you can enter a MACID as 00D071020A18, 00-D0-71-02-0A-18, or as 00 D0 71 02 0A 18. If you enter an incorrect Lock ID, a warning informing you that the Lock ID you entered is invalid will appear when you attempt to create a license.
  7. Accept the default **Secret Key** defined for the FPM in your license configuration file. You will need to specify the secret key defined for your FPM in the license validation routine that you will need to add to your FPM application. It is therefore recommended that you do not change the default secret key to ensure that you specify the correct secret key in the license validation routine.
  8. In the **Secure Algorithm** property, specify an index that corresponds to a security algorithm that is defined in the security DLL file. The sample security DLL file uses an algorithm that has an index of **0**. If you plan on using the sample security DLL file, you must specify **0** in this property or else the *i.LON License Generator* will not be able to generate an FPM license. See *Building the Security DLL File* for more information on using algorithm indexes.
  9. In the **Customer Information:** box, enter any pertinent customer data that you want recorded in your FPM application license log file such as the company name, company representative, address, phone number, and email address. Your FPM application license log file is updated each time you generate an FPM application license.
  10. Click **Create License**. A dialog opens in which you save the FPM license to an XML file. The default file name of the FPM license is `<ShortCompanyName><LonMarkID><ShortFeatureName>.xml`.



11. Specify the folder on your computer where customer FPM application licenses are to be saved and then click **Save**. By default, the FPM license is saved to the LonWorks\iLON\Development\Licensing\iLONLicenseGen folder. Once you save the FPM application license, your FPM application license log file is updated.
12. Optionally, you can view your updated FPM application license log file. To do this, browse to the LonWorks\iLON\Development\Licensing\iLONLicenseGen folder and then open the **iLONLicenses.log** file with a text editor.

```

*****
* License created on: Wednesday, September 05, 2007 13:34:42
* Created by: Our Corporation
* LonMark ID: 0
* Customer Information:
-----
Your Company

John Doe

123 ABC. St.

San Jose, CA 95126

* License:
<?xml version="1.0" encoding="utf-8"?>
<LicenseManager>
  <VersionMajor>1</VersionMajor>
  <VersionMinor>0</VersionMinor>
  <License>
    <CompanyName>Our Corporation</CompanyName>
    <FeatureName>FPM HVAC Controller</FeatureName>
    <LicenseType>Unlimited</LicenseType>
    <LockType>MACID</LockType>
    <LockId>00d071020A18</LockId>
    <Options></Options>
    <LicenseKey>8DFB1477E44DF955226715411D4E5524</LicenseKey>
  </License>
</LicenseManager>

```

The FPM application license log file lists the following information:

- The date on which the FPM application license was created.
- Your company's name and LonMark ID
- The customer information you entered in step 9.
- The FPM license data, which consists of your company's name, the name of the FPM licensed, the license type, lock type, lock ID, any options specified, and the license key generated for the FPM application by the security DLL file.

---

## Supplying FPMs to Customers

After you built a release version of your FPM application and have created FPM application licenses for your FPMs, you can supply your FPM applications to customers. When customers order an FPM application from your company, you need to provide the following files for them:

- Your company's FPM resource file set in which you created the user-defined functional profile template (UFPT) used by your FPM application. Your company's FPM resource file set should be stored in the LonWorks\Types\User\<YourCompany> folder on your computer, and it consists of .ENU, .fmt, .fpt, .ls, and .typ files. These are the files you generated with when you created the UFPT for your FPM with the NodeBuilder Resource Editor.

See Chapter 3, *Creating FPM Templates*, for more information on creating FPM templates and generating your company's FPM resource file set.

- The device interface (XIF) file (.xif extension) that you created for your FPM application. Your XIF should be stored in the destination folder that you specified when you generated the XIF with the *i.LON SmartServer 2.0* LonWorks Interface Developer Tool.

See Chapter 4, *Creating FPM Device Interface (XIF) Files*, for more information on creating model files and converting them to the XIFs with the *i.LON SmartServer 2.0* LonWorks Interface Developer Tool.

- The FPM executable module (with license validation enabled). This is the .app file that is created and updated when you compile your FPM application with the *i.LON SmartServer 2.0* Programming Tool. By default, your FPM application is stored in the

LonWorks\iLON\Development\eclipse\workspace.fpm\*<company program ID>.UFPT<FPT Name>*\Release folder on your computer, and it is named *<company program ID>.UFPT<FPT Name>.app*.

See Chapter 5, *Creating FPMs*, for more information on creating and compiling the FPM application. See *Enabling License Validation in an FPM Application* in this chapter for more information on protecting your FPM applications.

- The FPM application license. This is the .xml file you created with the *i.LON License Generator* that is used to protect your FPMs. By default, your FPM licenses are stored in the LonWorks\iLON\Development\Licensing\iLONLicenseGen folder on your computer.

See the *Creating FPM Application Licenses* section in this chapter for more information on creating this file.

In addition to supplying the required files to your customers, you should also provide instructions that explain how to install your FPMs on their SmartServers. The following is a set of sample instructions that you can use or modify:

1. Verify that an FPM programming license is installed on your SmartServer. If FPM programming is not licensed on your SmartServer, you can order a FPM programming license from the *i.LON SmartServer 2.0* Web site at [www.echelon.com/products/cis/activate](http://www.echelon.com/products/cis/activate).
2. Use FTP to access the root/lonworks/types folder on the flash disk of your SmartServer. Copy the supplied resource file set to the root/lonworks/types folder.
3. Use FTP to access the root/lonworks/import folder on the flash disk of your SmartServer. Copy the supplied device interface (XIF) file (.xif extension) to the root/lonworks/import folder.
4. Use FTP to access the root/config/license folder on the flash disk of your SmartServer. Copy the supplied FPM application license (.xml file) to the root/config/license folder (or a subfolder, if required by the license validation routine).
5. Use FTP to access the root/modules folder on the flash disk of your SmartServer. Copy the supplied FPM executable module (.app file) to the root/modules folder.
6. Deploy, test, and connect the FPM application following the instructions in Chapter 6, *Deploying FPMs on a SmartServer*.



# Localizing the SmartServer Web Interface

This chapter describes how to translate custom SmartServer Web pages and the entire SmartServer Web interface to a different language.

---

## Language Localization Overview

You can localize the language of the SmartServer Web interface using the *i.LON SmartServer 2.0 Programming Tool*. The SmartServer includes English, German, and French languages, but you can work with the SmartServer in any one-byte or two-byte character language by translating the **.properties** files on the SmartServer.

You can perform language localization using either the demo version of the *i.LON SmartServer 2.0 Programming Tools* included on the *i.LON SmartServer 2.0 DVD* or using the full version on the *i.LON SmartServer 2.0 Programming Tools* included on the *i.LON SmartServer 2.0 Programming Tools DVD*.

To localize the language of the SmartServer Web interface, you create a language localization project in the *i.LON SmartServer 2.0 Programming Tool*. You can then create localized custom SmartServer Web pages, or you can localize the entire SmartServer Web interface.

Creating localized custom SmartServer Web pages entails doing the following:

1. Translating the **COMMON.properties** file on the SmartServer flash disk with the *i.LON SmartServer 2.0 Programming Tool*.
2. Translating the **.properties** file of any embedded application that you plan on using in your custom SmartServer Web page. For example, if you wanted to create a custom Web page that contains an Event Scheduler, you would translate the **800001012800000[4].UFPTscheduler.properties** file with the *i.LON SmartServer 2.0 Programming Tool*.
3. Creating a new custom SmartServer Web page using *i.LON Vision 2.0*, adding application objects to the Web page, selecting the localized language as the default, and then publishing the custom SmartServer Web page.

Localizing the language of the SmartServer Web interface entails doing the following:

1. Translating one-by-one all of the **.properties** file in the `root/web/nls/echelon` folder on the SmartServer flash disk with the *i.LON SmartServer 2.0 Programming Tool*.
2. Creating a new language folder in the working copy of the SmartServer embedded image on your computer.
3. Editing the **index.htm** file with a text editor so that you can select your language from your *i.LON SmartServer 2.0*'s home page.
4. Translating and updating the language settings of the **Welcome.htm**, **Menu.htm**, **Sidebar.htm** files with *i.LON Vision 2.0*, or with a text editor.

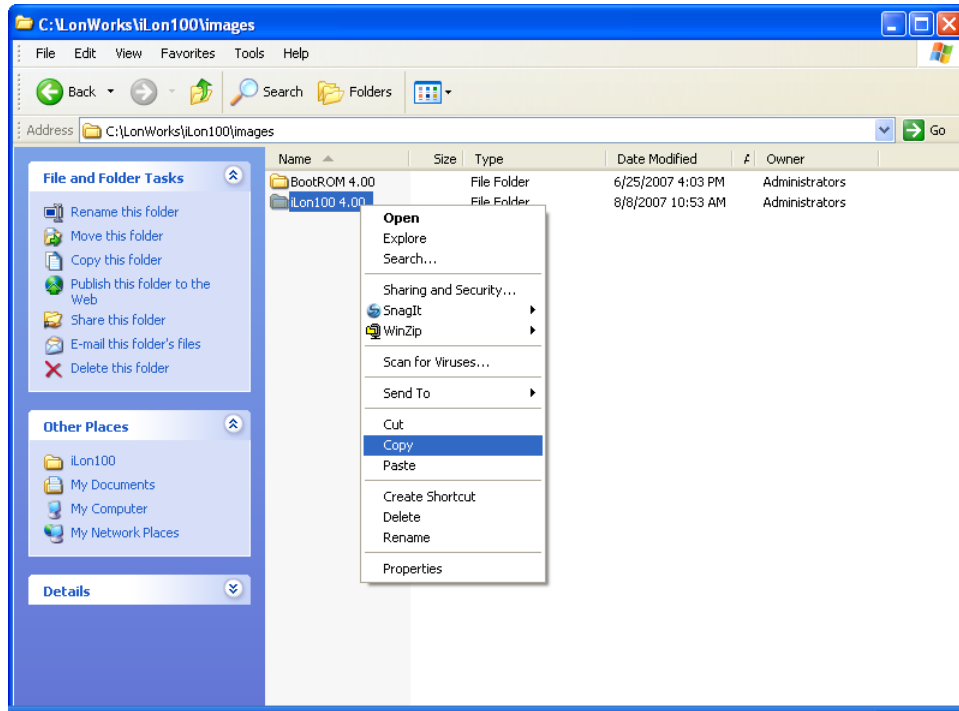
The following sections describe how to create a language localization project, how to create localized custom SmartServer Web pages, and how to translate the SmartServer Web interface.

---

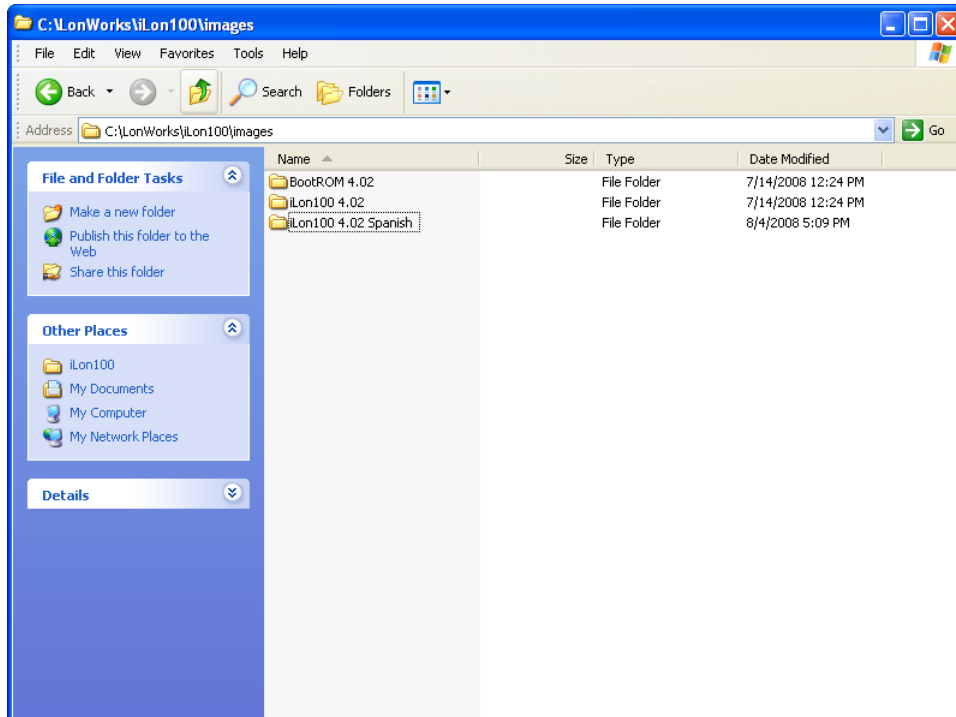
### *Creating a Language Localization Project*

You can create a new language localization project using either the demo or full version of the *i.LON SmartServer 2.0 Programming Tool*. To do this, follow these steps:

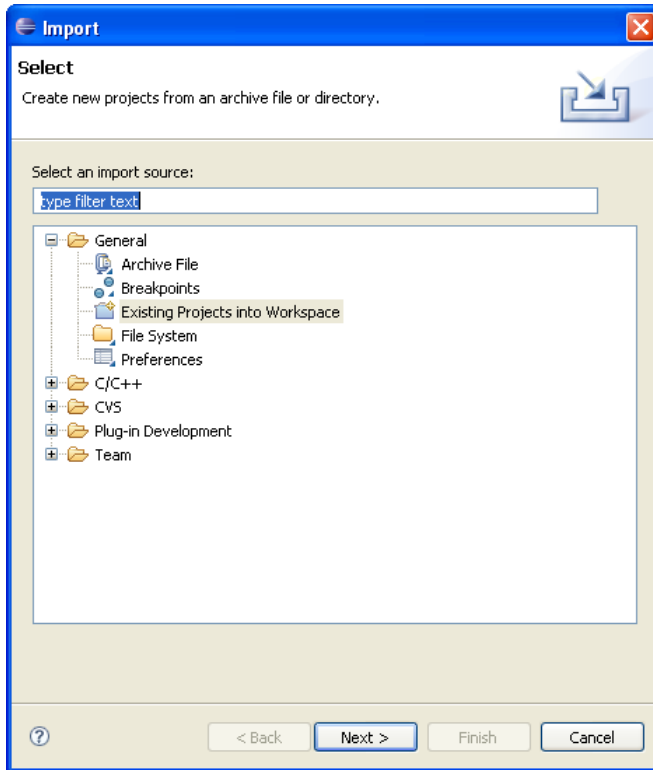
1. Verify that the SmartServer embedded image is installed in the `LonWorks\iLon100\images\iLon100 4.0<x>` directory on your computer. The embedded image is installed in this directory when you install the *i.LON SmartServer 2.0* software from the *i.LON SmartServer 2.0 DVD*. For more information on installing the *i.LON SmartServer 2.0* software, see the *i.LON SmartServer 2.0 User's Guide*.
2. Create a working copy of the SmartServer embedded image on your computer. To do this, follow these steps:
  - a. Copy the `iLon100 4.0<x>` folder in the `LonWorks\iLon100\images` directory on your computer.



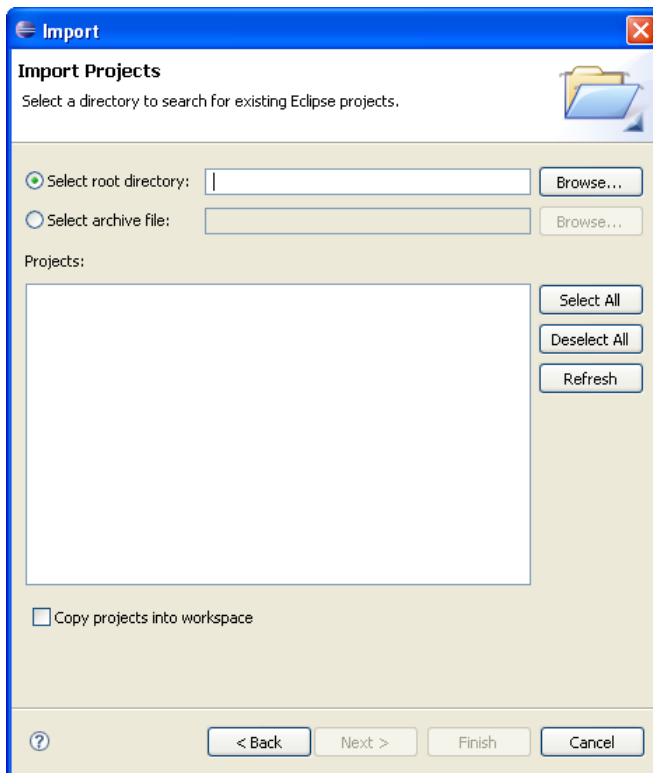
- b. Paste the folder in the same directory.
- c. Rename the folder to something meaningful such as “*i.LON 100 4.0<x> <Language>*”.



3. Start the *i.LON SmartServer 2.0 Programming Tool*. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0 Programming Tool* opens.
4. Click **File** and then click **Import**. The **Import** dialog opens in the Select window.

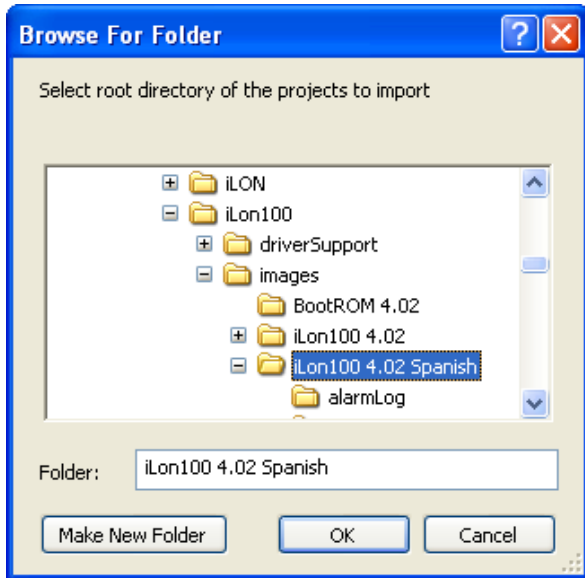


5. Expand the **General** folder, click **Existing Projects into Workspace**, and then click **Next**. The Import Projects window opens.

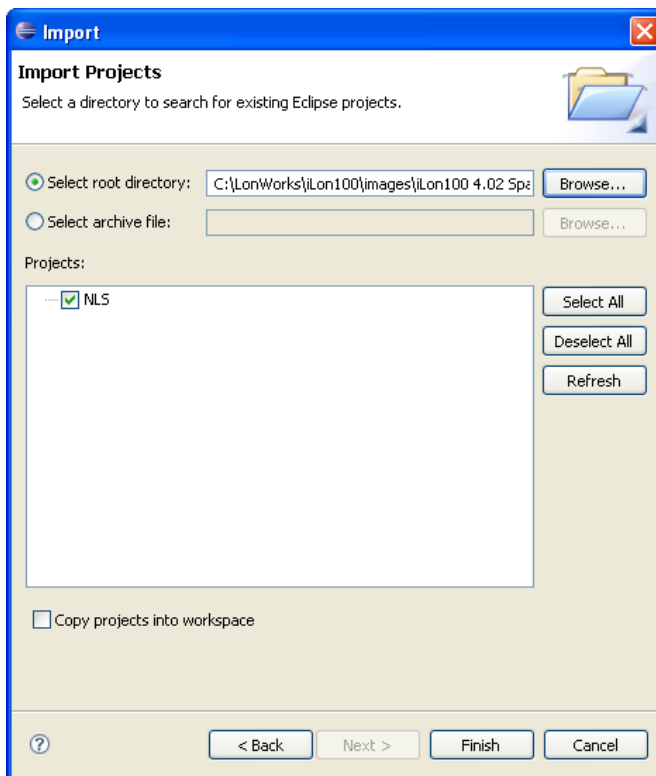


6. Click **Browse**. The **Browse to Folder** dialog opens.

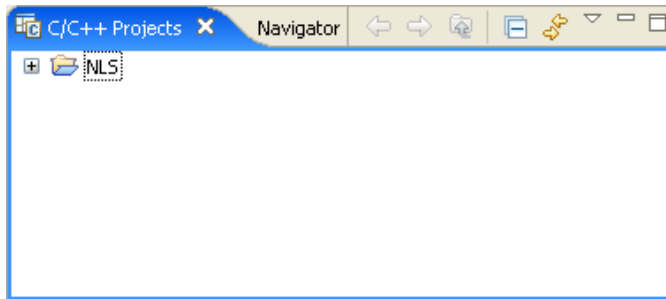




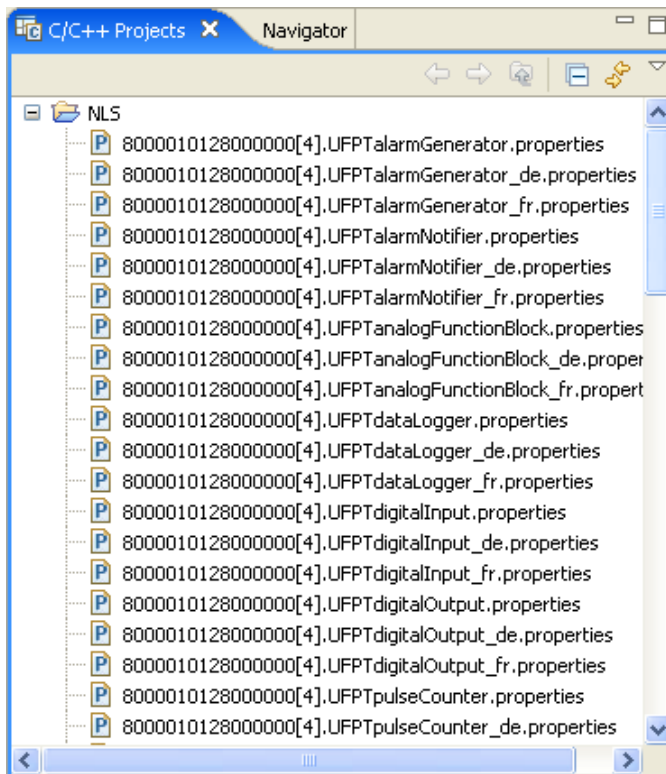
7. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language> folder and then click **OK**. A new project called **NLS** appears in the **Projects:** box. This means that your language localization project has been created within the current workspace.



8. Click **Finish**. An NLS project appears in the **C/C++ Projects** view.



- Expand the **NLS** folder. All the English, German, and French **.properties** files for the SmartServer embedded applications, system setup Web pages, and headers and properties appear under the **NLS** folder.



**Note:** A language localization project is stored in its own set of resource files; therefore the installation of an updated version of the SmartServer embedded image will not conflict with these resource files. After you install an updated SmartServer embedded image on your computer, you just need to copy it to the working copy of the embedded image you created in step 1. The *i.LON SmartServer 2.0 Programming Tool* will resolve any differences in your language localization project.

## *Creating Localized Custom SmartServer Web Pages*

You can localize the language for the new individual custom Web pages you are planning to build. To do this, you do the following:

- Translate the **COMMON.properties** file in the root/web/nls/echelon folder on the SmartServer flash disk with the *i.LON SmartServer 2.0 Programming Tool*.
- Translate the **.properties** file of any embedded application that you plan on using on in your custom SmartServer Web page. For example, if you wanted to create a custom Web page that contains the Scheduler object, you would translate the

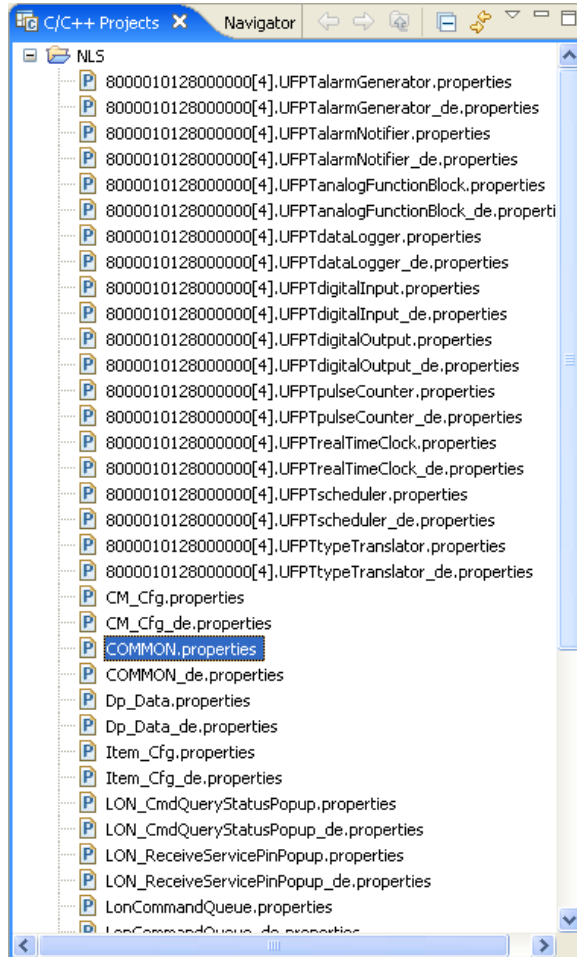
**8000010128000000[4].UFPTscheduler.properties** files in the root/web/nls/echelon/ folder with the *i.LON SmartServer 2.0 Programming Tool*.

3. Create a custom SmartServer Web page using *i.LON Vision 2.0*.

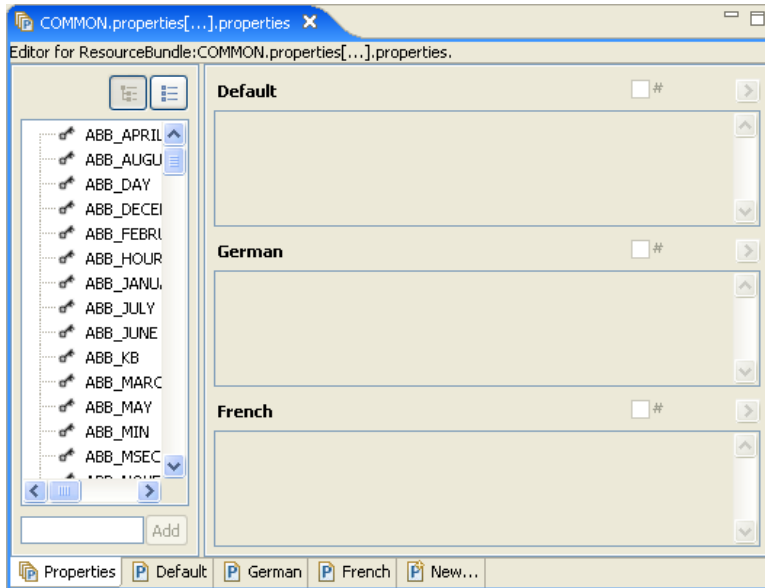
### *Translating Common Properties*

You can translate the **COMMON.properties** file in the root/web/nls/echelon/ folder on the SmartServer flash disk with the *i.LON SmartServer 2.0 Programming Tool*. To translate this file, you do the following:

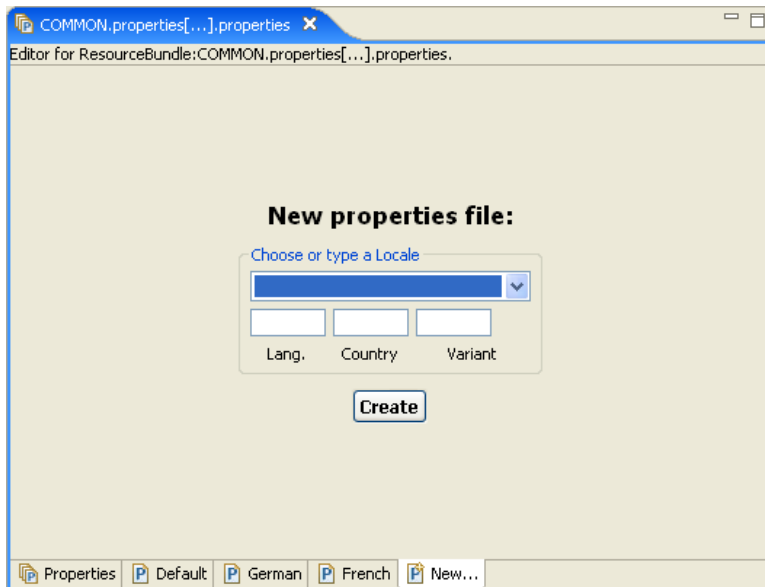
1. In the **C/C++ Projects** view of the *i.LON SmartServer 2.0 Programming Tool*, click the **COMMON.properties** file under the **NLS** project.



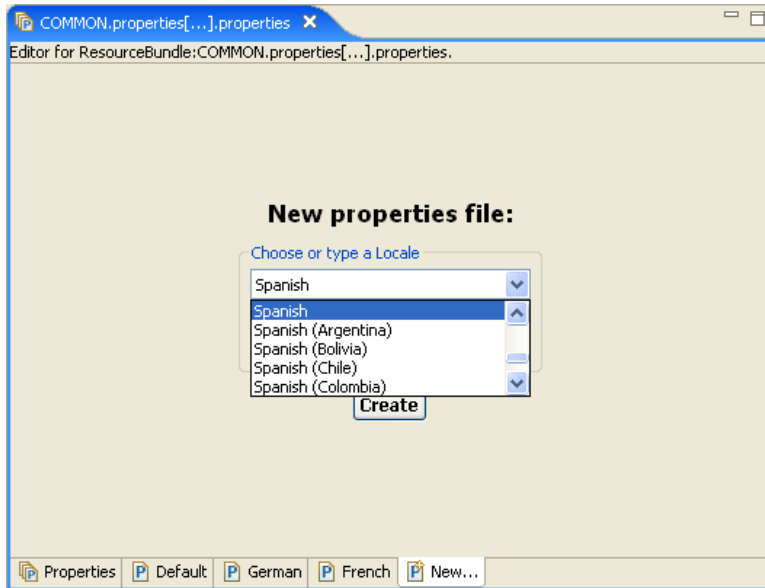
2. The **Properties Editor** view opens.



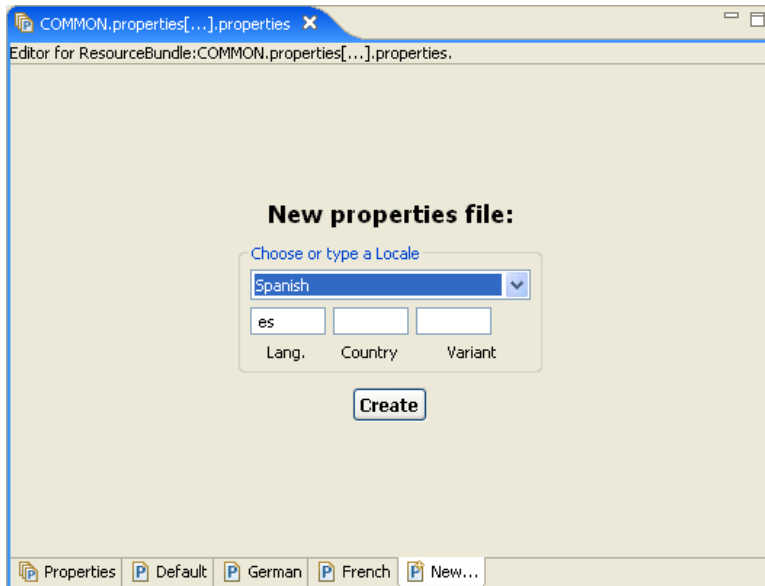
3. The left frame in the view lists all the common properties in the SmartServer Web interface. The right frame includes boxes that display the English (**Default**), **German**, and **French** translations of a selected property. The bottom includes tabs that you can click to view and edit a list of all the properties within a **.properties** file for a specific language.
4. Click the **New** tab at the bottom of the **Editor** view. The **New Properties File:** dialog opens.



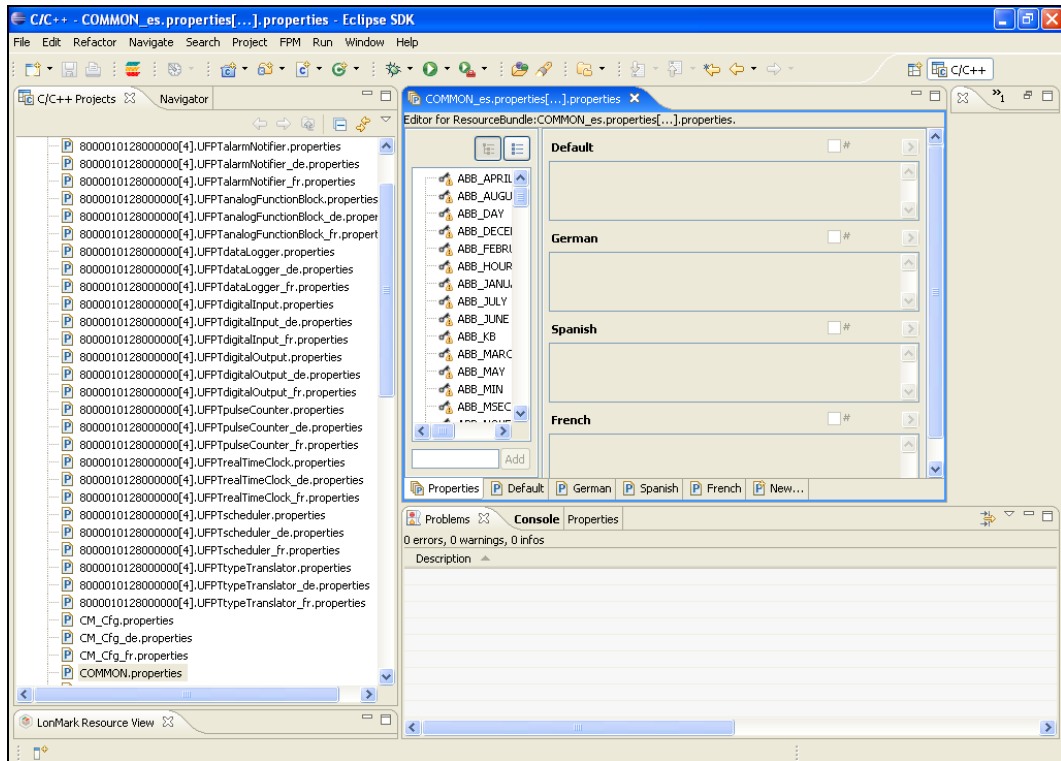
5. In the **Choose or Type Locale** box, select the language and desired version (if different regional varieties are available for the language) to which your custom SmartServer Web page is to be translated.



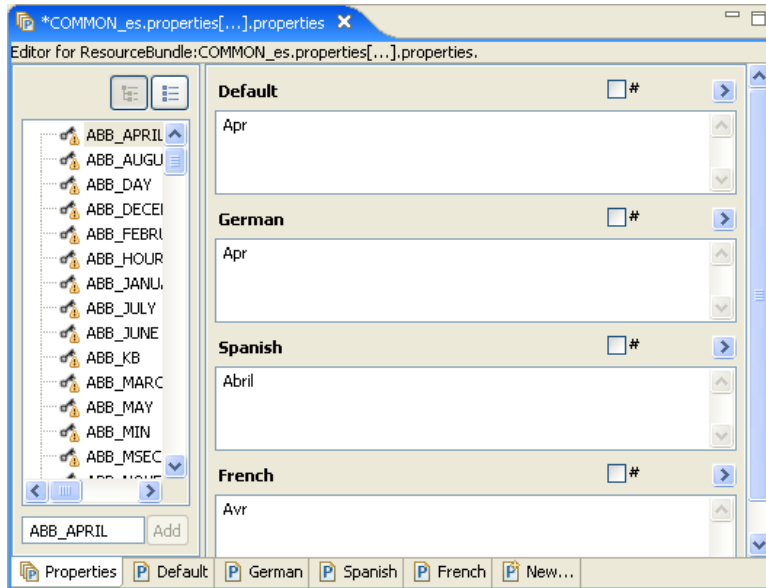
6. The **Lang.** and **Country** properties are filled in. Optionally, you can enter a **Variant** to further categorize the selected language. This is useful if you want to create different translations of the same language with the same regional version.



7. Click **Create**.

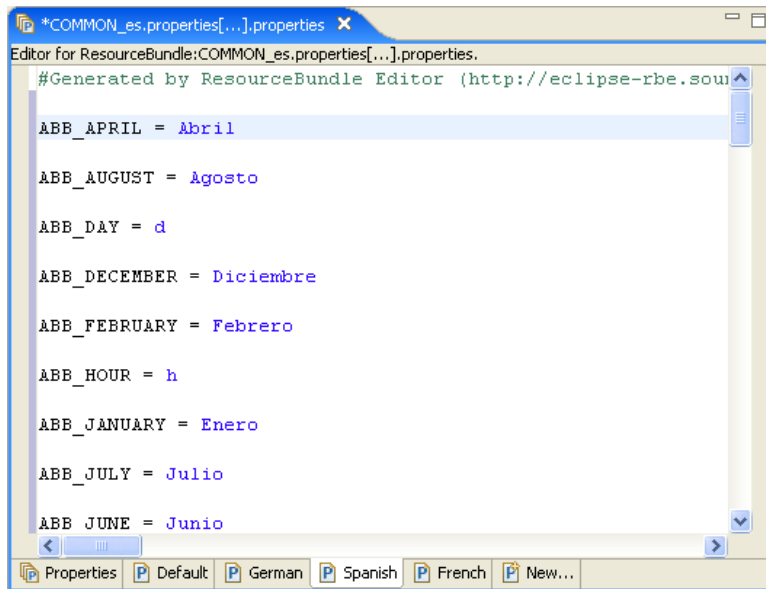


8. A new **COMMON<\_language[\_REGION] [\_variant]>.properties** file is added to the **C/C++ Projects** view and this file appears in the **Editor** view. In addition, a box marked with the language you selected is added to the bottom of the right frame of the **Editor** view. Note that all the properties listed in the left frame are marked with warning symbols, indicating that the property has not yet been translated. Once you enter a translation for a property, the warning symbol is removed.
9. Translate each property listed in the left frame. You can do the translation from the **Properties** tab (recommended) or from your language's tab.
  - To translate the properties from the **Properties** tab, click each property listed in the left frame and enter a translation in your language box in the right-pane one-by-one. This is the slower approach as you must enter text for properties that do not have translations for the **Default** (English) language (for example, the abbreviations for units of time and the abbreviations for some months), and you are repeatedly clicking in between typing.



You can comment out the text in a translation by selecting the checkbox ( #) in the upper right-side of the language box. You can switch to your language's property tab by clicking the arrow () on the upper right-hand corner of your language box.

- To translate the properties from your language's tab, first copy the **Default** (English) translation and paste it into your language's tab. You can begin translating the properties listed in your language's tab.



**Tip:** Save your language localization project frequently to safeguard your data from a power outage or other failure. To save your language localization project, click **File** and then click **Save**.

10. When you finished translating all the properties in the **COMMON.properties** file, save your language localization project.
11. Copy the localized copy of the **COMMON.properties** file to the SmartServer. To do this, follow these steps:

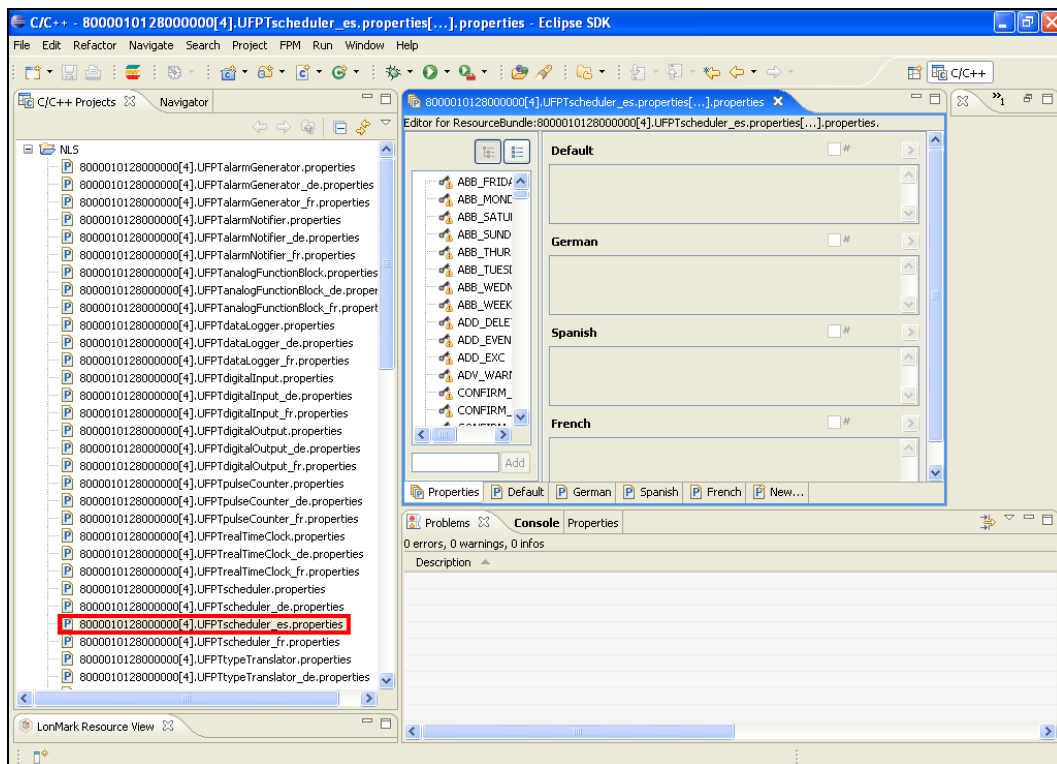
- Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\nls\echelon directory or on your computer (or other location where your working copy of the NLS files is stored).
- Use FTP to access the root/web/user/echelon/folder on the flash disk of your SmartServer.
- Copy the **COMMON<\_language[\_REGION] [\_variant]>.properties** file in the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\nls\echelon folder on your computer to the root/web/nls/echelon/ folder on the SmartServer flash disk.

### Translating Embedded Application Properties

You can implement SmartServer embedded applications (Event Scheduler, Data Logger, Alarm Notifier, and so on) in your custom SmartServer Web pages and have the properties in the applications appear in a localized language. To do this, you translate the .properties file of any embedded application that you plan on using in your custom SmartServer Web page.

For example, if you wanted to create a custom Web page that contains an Event Scheduler in a localized language, you would translate the **8000010128000000[4].UFPTscheduler.properties** file.

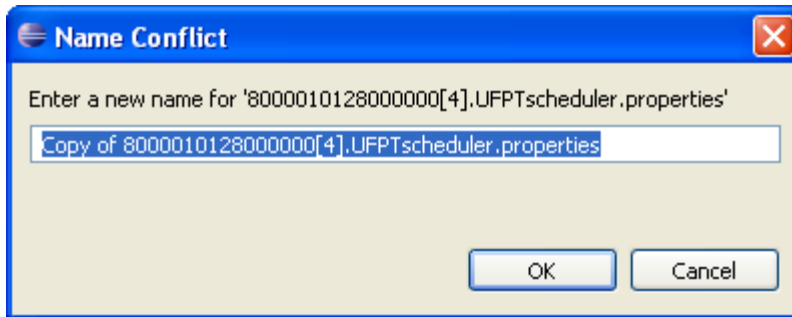
To translate this file, you would essentially follow the steps outlined in the previous section, *Translating the COMMON.properties File*, except that in step 1, you click the **8000010128000000[4].UFPTscheduler.properties** file under NLS project in the **C/C++ Projects** view.



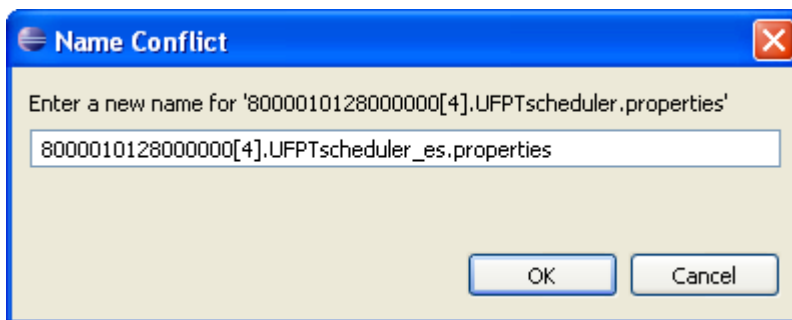
**Tip:** Alternatively, you can create a new localized .properties file for an embedded application from a copy of an existing English, German, or French version. To do this, you do the following,

- Copy and paste the existing English, German, or French version of the .properties file of the embedded application in the **C/C++ Projects** view. The **Name Conflict** dialog opens.





2. Re-name the copy by deleting the “Copy of” pre-fix and inserting the “<\_language[\_REGION] [\_variant]>” suffix between the name of the embedded application and the **.properties** extension.  
 For example, you can create a Spanish version of the Event Scheduler by copying and pasting the **8000010128000000[4].UFPTscheduler.properties** file and re-naming it **8000010128000000[4].UFPTscheduler\_es.properties**.




3. Click **OK**. The new localized version of the **.properties** file appears in the **C/C++ Projects** view.
4. Double-click the new localized version of the **.properties** file to begin translating its properties in the **Editor** view.
5. Copy the localized copy of the embedded application’s **.properties** file to the SmartServer. To do this, follow these steps:
  - a. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\nls\echelon directory or on your computer (or other location where your working copy of the NLS files is stored).
  - b. Use FTP to access the root/web/user/echelon/ folder on the flash disk of your SmartServer.
12. Copy the <application><\_language[\_REGION] [\_variant]>**.properties** file in the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\nls\echelon folder on your computer to the root/web/nls/echelon/ folder on the SmartServer flash disk.

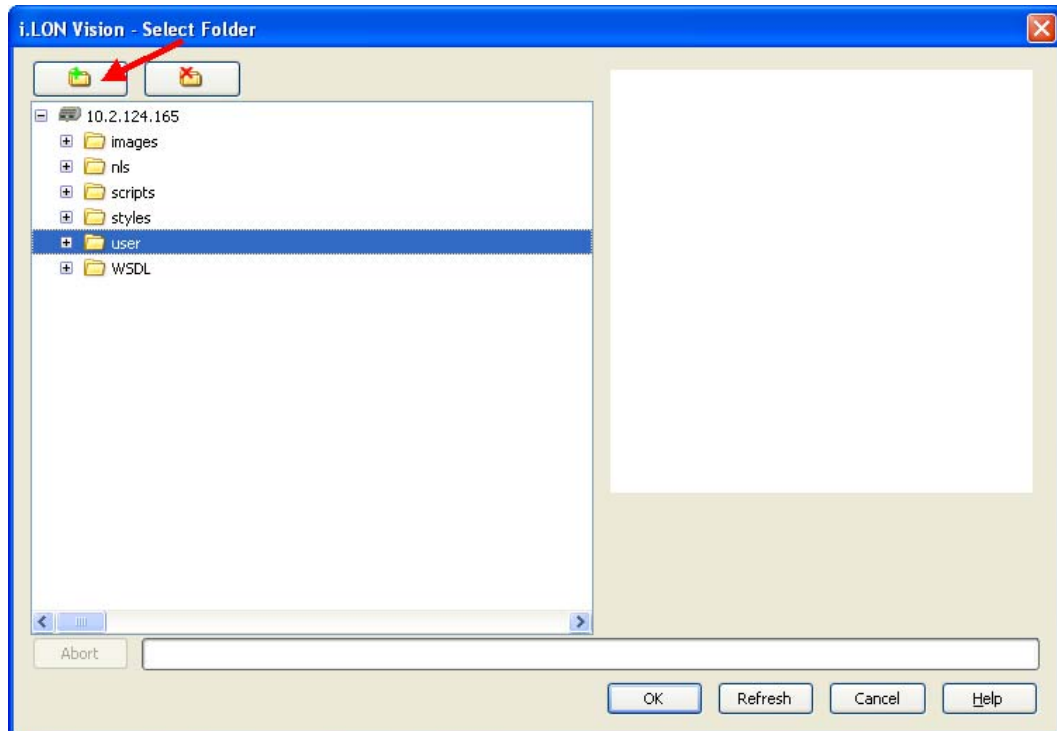
### *Creating a Localized Custom SmartServer Web Page*

You can create new custom SmartServer Web pages using *i.LON Vision 2.0* and have the Web pages appear in a localized language. To create localized custom SmartServer Web pages, you must translate the **COMMON.properties** file and the **.properties** file of the application objects to be used in your custom SmartServer Web pages as described in the previous sections. In addition, *i.LON Vision 2.0* must be installed on your computer. For more information on installing *i.LON Vision 2.0*, see the *i.LON Vision 2.0 User’s Guide*.

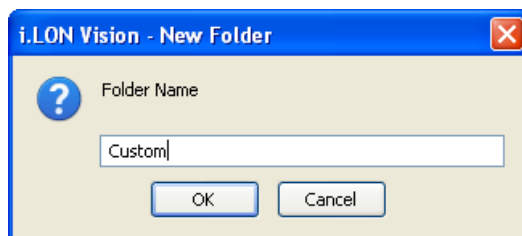
To create a localized custom SmartServer Web page, you do the following:

1. Start *i.LON Vision 2.0*. To do this, click **Start**, point to **Programs**, point to Echelon **i.LON Vision 2.0 SmartServer 2.0**, and then click **i.LON Vision 2.0 SmartServer 2.0**. *i.LON Vision 2.0* opens.

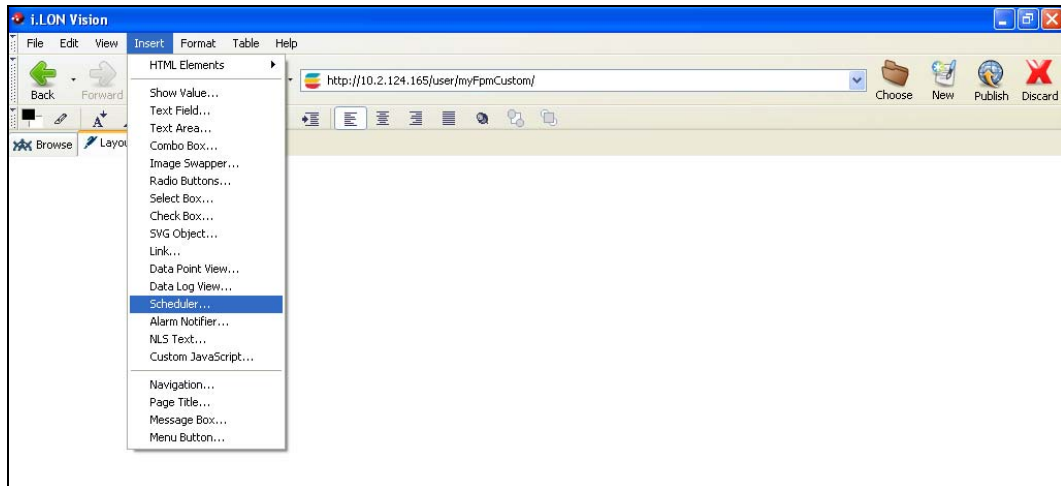
2. Connect *i.LON Vision 2.0* to your SmartServer. To do this, click **Manage Connections** in the Sites pane on the left side, or click **File** and then click **Site Manager**. The **Site Manager** dialog opens. Click **New Site**, the **Edit Site** dialog opens. Enter your SmartServer's information, and then click **OK** twice. A link with the IP address of your SmartServer is added to the Sites pane.
3. Create a new custom SmartServer Web page. To do this, click the **New** button on the Editor toolbar () or click **File** and then click **New Page**. The **Select Folder** dialog opens.
4. Expand the SmartServer icon, expand and click the **user** folder (you must create the new folder in the root/web/user directory on the SmartServer flash disk), and then click the New Folder icon to create the directory for your custom SmartServer 2.0 Web page.



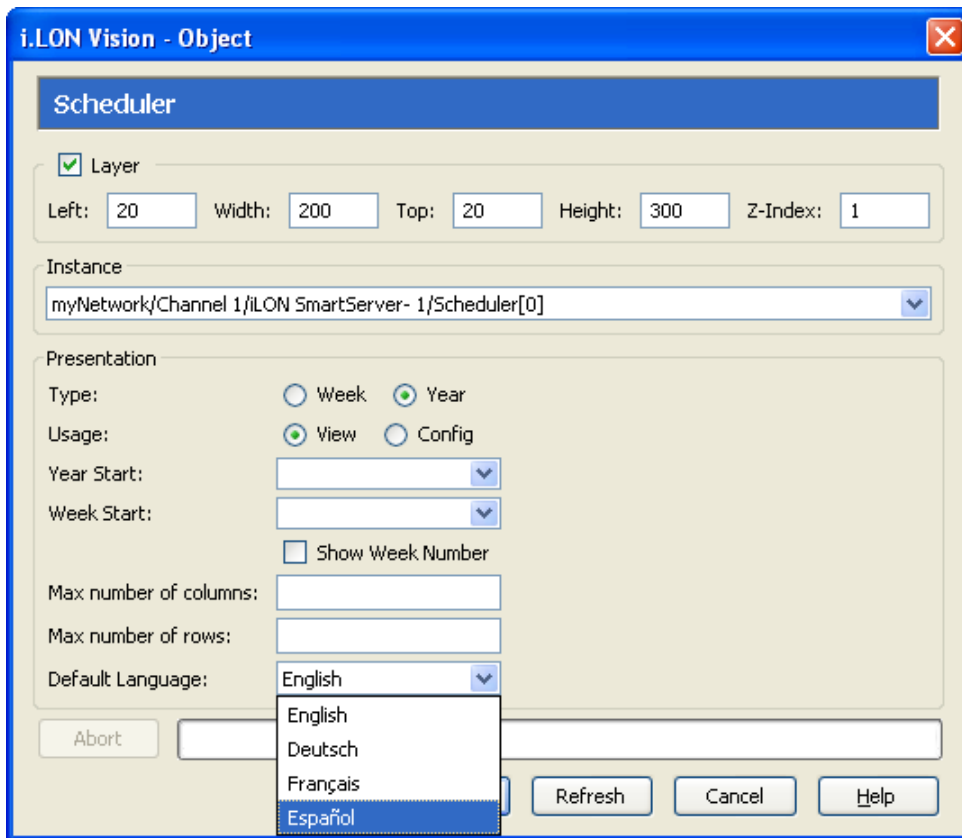
5. The **New Folder** dialog opens. Enter the name for the folder where all the custom SmartServer 2.0 Web pages for a given Web design will be stored. Click **OK**.



6. Click the custom SmartServer 2.0 Web page folder you created in step 5, and then click **OK**.
7. In your new custom SmartServer Web page, click **Insert**, and then select one of the following objects that represent the application objects you can add to your custom SmartServer Web page: **Data Point View**, **Data Logger View**, **Scheduler**, or **Alarm Notifier**. This example uses a Scheduler object.



8. The **iLON Vision -Object** – <Application> dialog opens.
9. Configure the application object to fit the functionality provided by your custom SmartServer Web page. In the **Default Language** box, select your localized language, and then click **OK**.

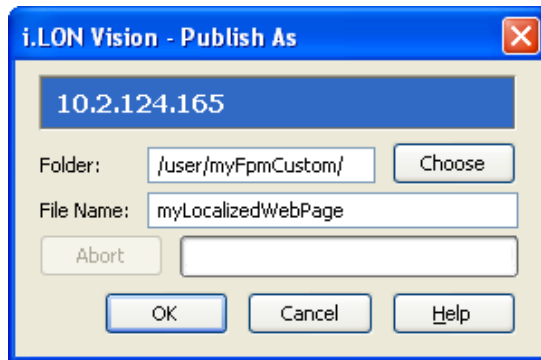


10. Edit and link your custom SmartServer Web page as described in the *iLON Vision 2.0 User's Guide*.



11. Click **Publish** on the Editor toolbar (Publish), click the **Browse** tab, or click **File** and then click **Publish**. The **Publish As** dialog opens.

12. In the **File Name** property, enter the name of the **.htm** file (one word with no spaces), and then click **OK**.

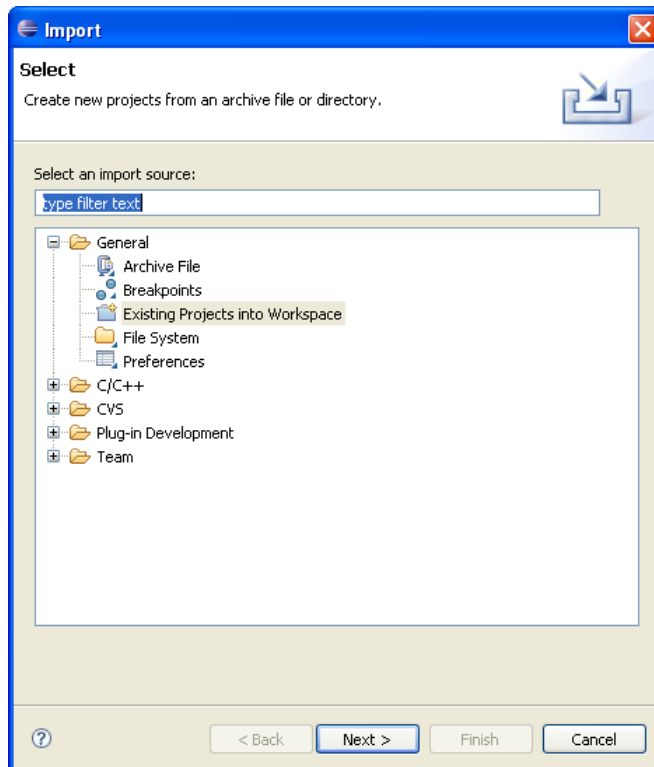


---

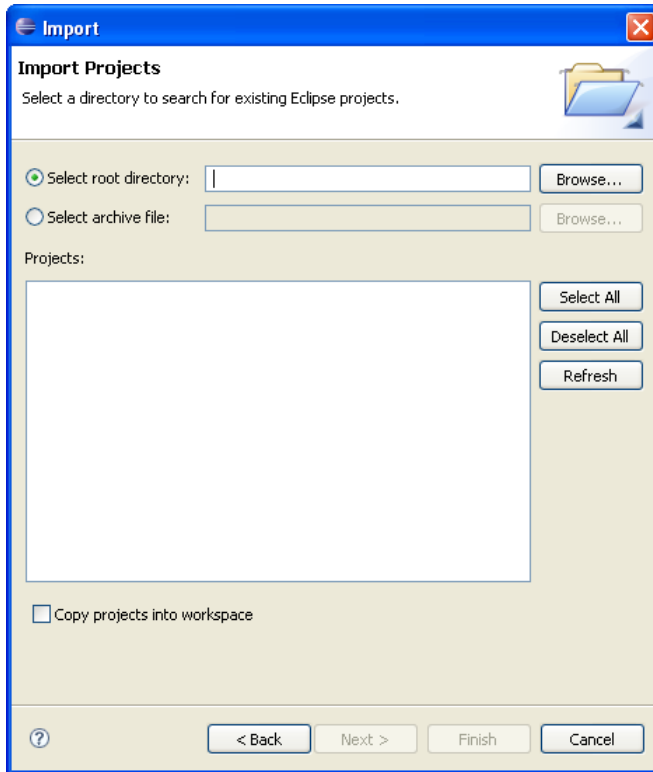
## *Creating Localized FPM Configuration Web Pages*

You can localize the language for custom FPM configuration Web pages. To do this, follow these steps:

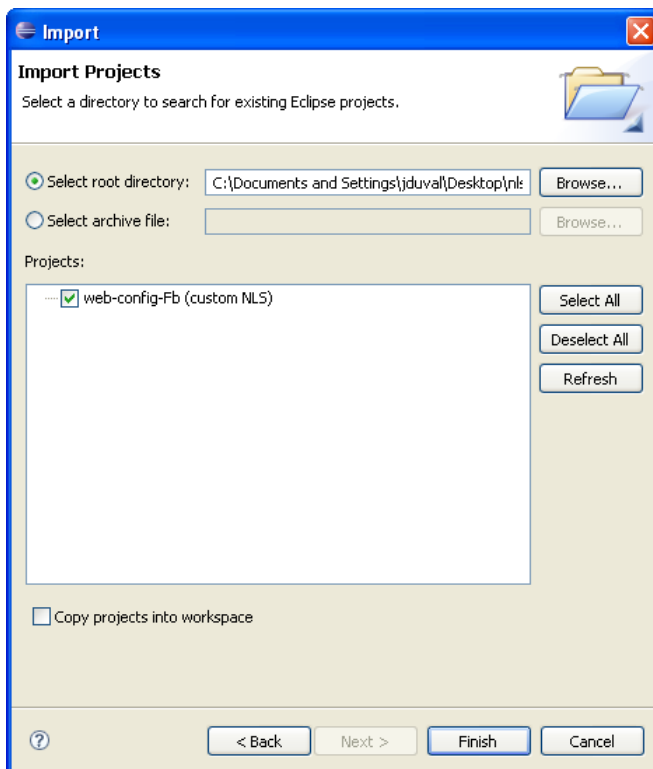
1. Copy the root/web/config/Fb/NLS folder on the SmartServer flash disk to your computer.
2. Start the *i.LON SmartServer 2.0 Programming Tool*. To do this, click **Start**, point to **Programs**, point to **Echelon i.LON SmartServer 2.0 Programming Tools**, and then click **i.LON SmartServer 2.0 Programming Tools**. The *i.LON SmartServer 2.0 Programming Tool* opens.
3. Click **File** and then click **Import**. The **Import** dialog opens in the Select window.



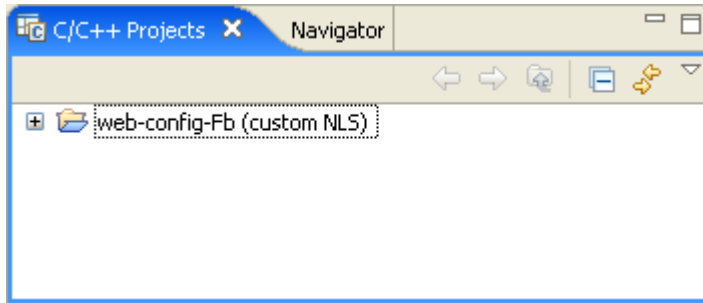
4. Expand the **General** folder, click **Existing Projects into Workspace**, and then click **Next**. The **Import Projects** window opens.



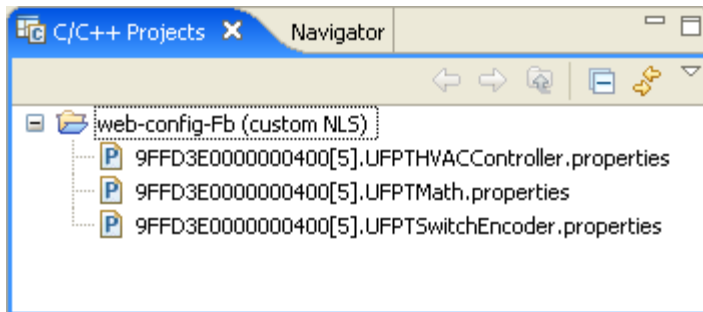
5. Click **Browse**. The **Browse to Folder** dialog opens.
6. Browse to the web/config/Fb/NLS folder you copied to your computer and then click **OK**. A new project called **web-config-Fb (custom NLS)** appears in the **Projects:** box. This means that your language localization project has been created within the current workspace.



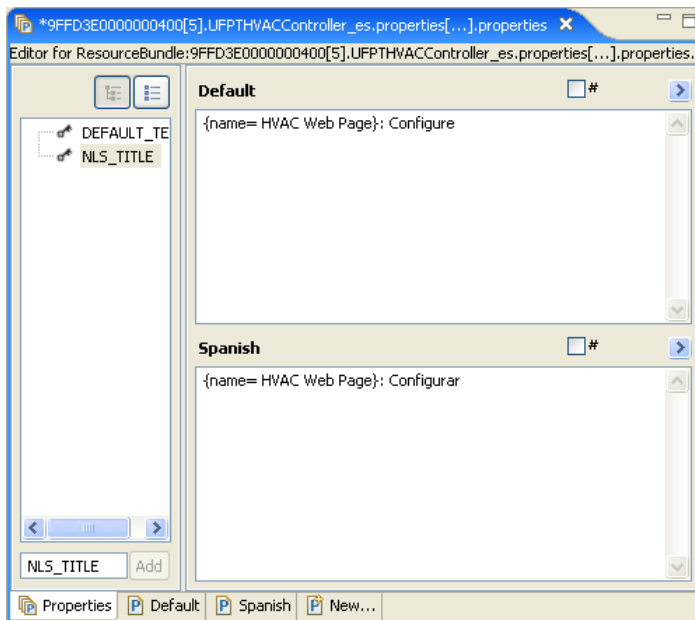
7. Click **Finish**. A **web-config-Fb (custom NLS)** project appears in the **C/C++ Projects** view.



8. Expand the **web-config-Fb (custom NLS)** folder. The English **.properties** files for the FPM configuration Web pages appear under the **NLS** folder.



9. Create a new **.properties** file for the FPM configuration Web page as described in *Translating Common Properties* earlier in this chapter.
10. Translate any **NLS Text** objects you added to your FPM configuration Web page and translate the page title property (NLS\_TITLE). This property provides the page title for your localized FPM configuration Web page. Do not modify the text that is enclosed in braces. For example, the following page title “{name=HVAC Web Page}: Configure”, could be translated to “{name=HVAC Web Page}: Configurar” in Spanish. For more information on **NLS Text** objects and how to change the page title for custom FPM Configuration Web pages, see *Creating Custom FPM Configuration Web Pages*.



---

## Localizing the Language of the SmartServer Web Interface

Localizing the language of the SmartServer Web interface entails doing the following:

1. Translating one-by-one all of the **.properties** file in the web/nls/echelon folder on the SmartServer flash disk with the *i.LON SmartServer 2.0 Programming Tool*.
2. Creating a new web/user/echelon/<language[\_REGION] [\_variant]> folder from a copy of the existing web/user/echelon/de (German) folder or the web/user/echelon/fr (French) folder in the working copy of the SmartServer embedded image on your computer.
3. Editing the **index.htm** file in the web folder with a text editor so that you can select your language from your *i.LON SmartServer 2.0*'s home page.
4. Translating the **Welcome.htm** file in the web/user/echelon/<language[\_REGION] [\_variant]> folder with *i.LON Vision 2.0*, or with a text editor.
5. Translating and updating the language settings of the **Menu.htm** file in the web/user/echelon/<language[\_REGION] [\_variant]> folder with *i.LON Vision 2.0*, or with a text editor.
6. Updating language settings of the **Sidebar.htm** files in the web/user/echelon/<language[\_REGION] [\_variant]> folder with *i.LON Vision 2.0*, or with a text editor.
7. Viewing the results of your language localization project with the SmartServer Web interface.

### Translating Property Files

You can translate the **.properties** file on the SmartServer with the *i.LON SmartServer 2.0 Programming Tool*. To do this, you one-by-one create localized copies of the **.properties** files listed in the **C/C++ Projects** view and translate all the properties listed in the files.

The SmartServer contains a total of 24 **.properties** files in the web/user/echelon folder, consisting of 10 files for the embedded applications and 14 for the system setup pages and general properties. Each **.properties** file contains anywhere from 5 to 412 properties. There is a total of approximately 1,450 properties. You can use these figures in estimating the man hours required to complete a language localization project for the SmartServer.

If you want to translate the SmartServer **Help.htm** files in the web/user/echelon folder, you should first evaluate whether you have the resources requires for this task. Translating the **Help.htm** files requires an extensive effort (much greater than that required for the translation of the **.properties** files). Furthermore, the translation of the **Help.htm** files is not supported—you cannot use the *i.LON SmartServer 2.0 Programming Tool* to perform the translations. Instead, you need to use a text editor such as Notepad, WordPad, or TextPad if you want to translate the SmartServer **Help.htm** files.

When you have finished translating all the localized copies of the **.properties** file, save the language localization project, and then copy all the **.properties** files to the SmartServer following step 11 in *Translating Common Properties* in this chapter.

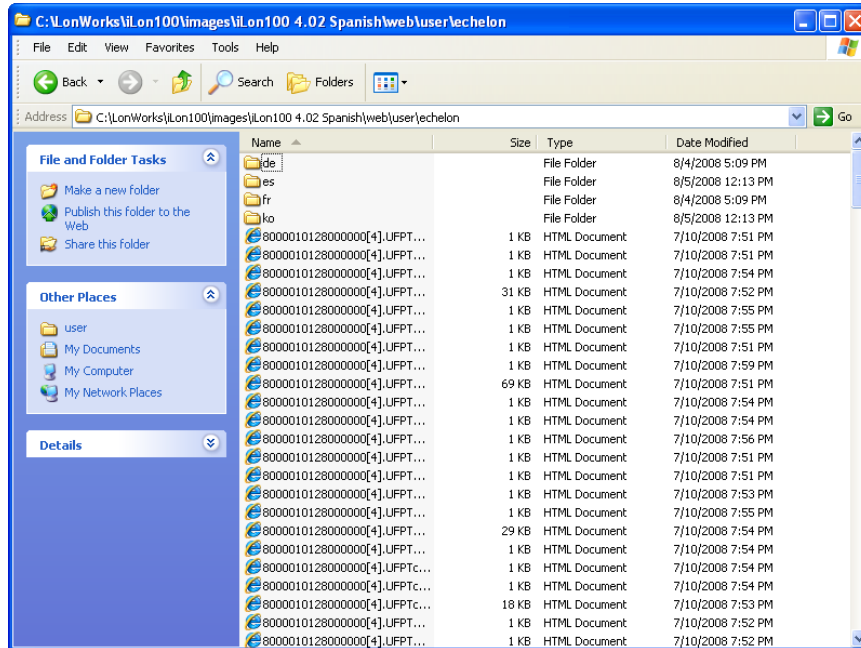
**Tip:** In addition to saving your language localization project frequently, you should regularly make backups of the web/nls folder in the working copy of the SmartServer embedded image on your computer. It is recommended that you make a backup each time you finish translating a file.

### Creating New Language Folders

You can create a new web/user/echelon/<language[\_REGION] [\_variant]> folder in the working copy of the SmartServer embedded image on your computer. You need to do this in order to create localized versions of the **Welcome.htm**, **Menu.htm**, and **Sidebar.htm** files. You will translate the text of the **Welcome** Web page in the **Welcome.htm** file and the menus and menu items in the **Menu.htm** file. In addition, you will change the language settings to your localized language in the **Menu.htm** and **Sidebar.htm** files.

To create a new web/user/echelon/<language[\_REGION] [\_variant]> folder, follow these steps:

1. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\user\echelon folder on your computer (or other location where your working copy of the SmartServer embedded image is stored).
2. Create a new folder named <language[\_REGION] [\_variant]>. For example, if you are creating a Korean translation, create a new folder named “ko”. If you are creating a Spanish translation, create a new folder named “es”.



3. Copy the **index.htm**, **Menu.htm**, **Sidebar.htm**, **Welcome.htm** files to the new <language[\_REGION] [\_variant]> folder.
4. Copy the <language[\_REGION] [\_variant]> folder to the SmartServer. To do this, follow these steps:
  - a. Use FTP to access the root/web/user/echelon folder on the SmartServer flash disk.
  - b. Copy the <language[\_REGION] [\_variant]> folder on your computer to the root/web/user/echelon folder on the SmartServer flash disk.

### *Editing the index.htm File to Enable a New Language on the SmartServer*

You can edit the **index.htm** file in the web folder with a text editor so that you can select your language from your i.LON SmartServer 2.0’s home page. After you enable your localized language in the **index.htm** file, you can copy the file to the SmartServer. To do this, follow these steps:

1. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web folder on your computer (or other location where the **index.htm** file in your working copy of the SmartServer embedded image is stored).
2. Open the **index.htm** file with a text editor such as Notepad, WordPad, or TextPad.
3. Locate the “<option>” elements on lines 157–159. Insert an <option> element for your localized language. For example, if you are creating a Spanish translation, insert the following text:

```
<option value="es/"> Español</option>
```



```

148 <body>
149 <div id="pagebody">
150 <div id="pageheader">
151 <span id="productlogo"><a href="http://www.echelon.com/ilon">
153 <table border="0" cellspacing="0" cellpadding="0">
154 <tr>
155 <td width=20> </td>
156 <td>Configuration &amp; Service: <select name="serviceSelect" id="servi
157 <option value="">English</option>
158 <option value="/de">Deutsch</option>
159 <option value="/fr">Français</option>
160 <option value="/es">Español</option>
161 </select></td>
162 <td width=20> </td>
163 <td width=64 id="login" onclick="navService();">Login</td>
164 </tr>
165 </table>
166 </div>
167 <div id="pageContent">
168 <table width="720" border="0" align="center" cellpadding="3">
169 <tr>
170 <td>
172 <ul id="product_points">
173 <li>Programmable, run custom C &amp; C++ applications and drive
174 <li>Direct LNS® interface to view and use the LNS database</li>
175 <li>Standalone mode can manage up to 200 PL &amp; 64 FT devices
176 <li>Network management using the SmartServer Web pages</li>
177 <li>More intuitive user interface</li>
178 <li>Create trend graphs using the configuration Web pages or i
179 <li>Localize configuration pages to any language</li>
180 <li>64MB of Flash and RAM</li>
181 </ul>
182 </td>
183 </tr>

```

4. Save the **index.htm** file.
5. Copy the **index.htm** file to the SmartServer. To do this, follow these steps:
  - a. Browse to the LonWorks\iLon100\images\iLon100 4.0<i> <Language>\web folder on your computer (or other location where the **index.htm** file in your working copy of the SmartServer embedded image is stored).
  - b. Use FTP to access the root/web folder on the SmartServer flash disk.
  - c. Copy the **index.htm** file on your computer to the root/web folder on the SmartServer flash disk.

### Translating the Welcome.htm File


You can translate the **Welcome.htm** file in the web/user/echelon/<language[\_REGION] [\_variant]> folder in your working copy of the SmartServer embedded image. You can do the translation with i.LON Vision 2.0, or you can do it with a text editor.

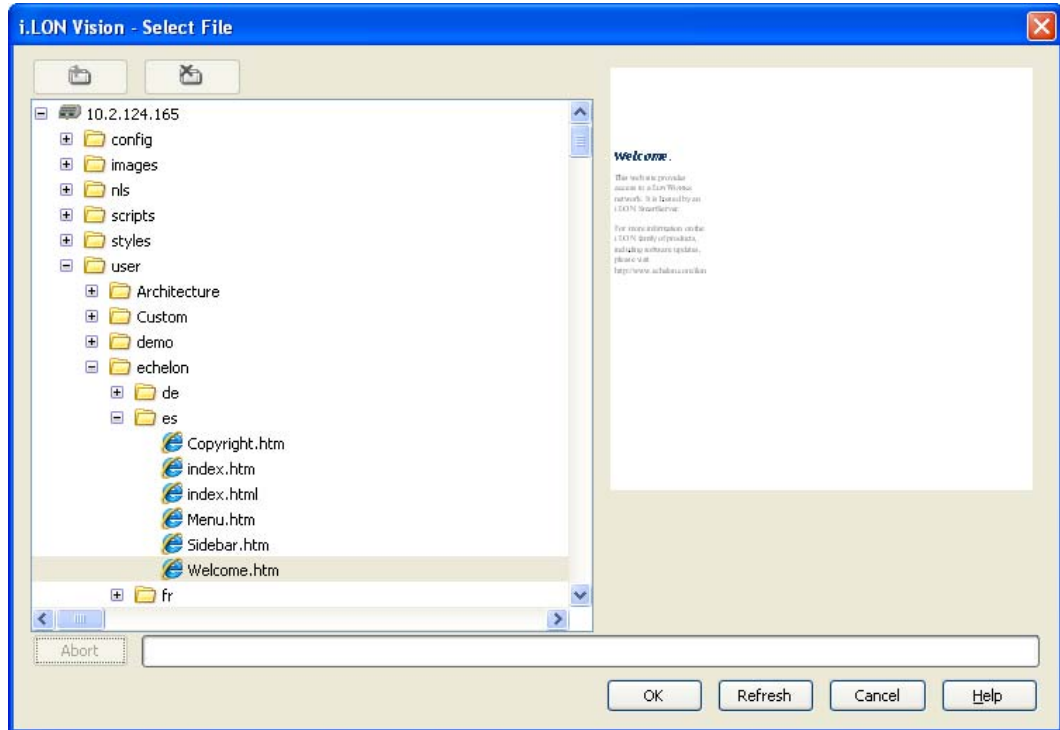
### Translating the Welcome.htm File with i.LON Vision 2.0

You can translate the **Welcome.htm** file using i.LON Vision 2.0. To do this, follow these steps:

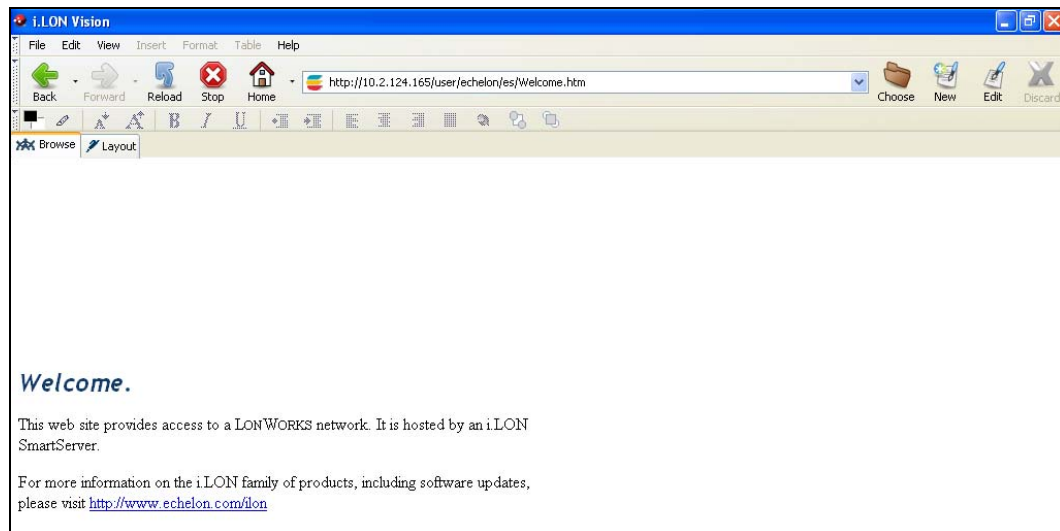
1. Copy your web\user\echelon\<language[\_REGION] [\_variant]> folder on your computer to the SmartServer. To do this, follow these steps:
  - a. Browse to the LonWorks\iLon100\images\iLon100 4.0<i> <Language>\web\user\echelon folder on your computer (or other location where your working copy of the SmartServer embedded image is stored).
  - b. Use FTP to access the root/web/user/echelon folder on the SmartServer flash disk.
  - c. Copy the <language[\_REGION] [\_variant]> folder on your computer to the root/web/user/echelon folder on the SmartServer flash disk.
2. Start i.LON Vision 2.0. To do this, click **Start**, point to **Programs**, point to Echelon **i.LON Vision 2.0 SmartServer 2.0**, and then click **i.LON Vision 2.0 SmartServer 2.0**. i.LON Vision 2.0 opens.
3. Connect i.LON Vision 2.0 to your SmartServer. To do this, click **Manage Connections** in the Sites pane on the left side, or click **File** and then click **Site Manager**. The **Site Manager** dialog


opens. Click **New Site**, the **Edit Site** dialog opens. Enter your SmartServer's information, and then click **OK** twice. A link with the IP address of your SmartServer is added to the Sites pane.

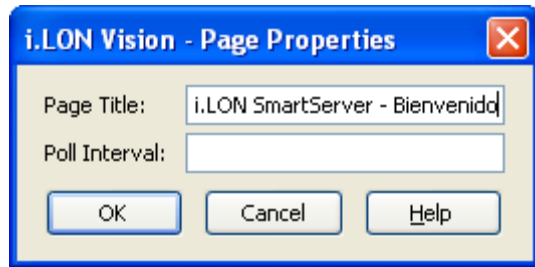
4. Click the Choose button () on the Editor toolbar to open the **Select File** dialog opens. Select the web/user/echelon/<language[\_REGION] [\_variant]>/Welcome.htm Web page, and then click **OK**.



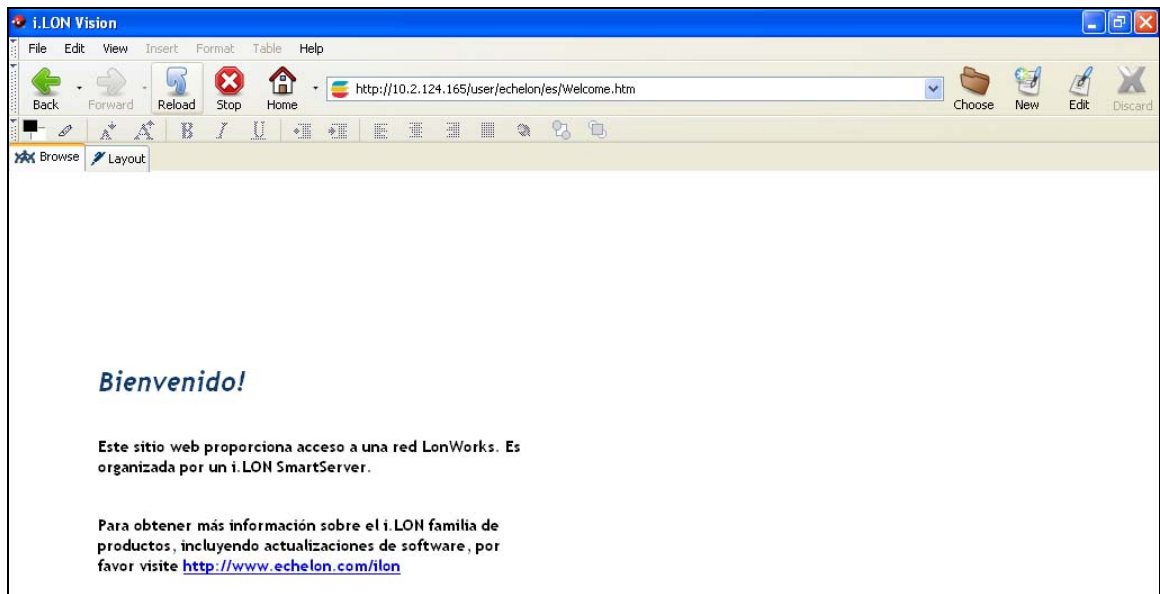
5. The English version of the **Welcome.htm** Web page opens.



6. Click **Edit** on the Editor toolbar () or click the **Layout** tab. Translate the text on the Web page from English to your localized language.
7. Translate the Page title following these steps:
  - a. Click **Format** and then click **Page Title Properties**. The **Page Properties** dialog opens.



- a. In the **Page Title** box, enter a descriptive page title.
  - b. Click **OK**.
8. Click **Publish**. The **Welcome.htm** Web page appears in your localized language.



### Translating the Welcome.htm File with a Text Editor

You can translate the **Welcome.htm** file using a text editor such as Notepad, WordPad, or TextPad. To do this, follow these steps:

1. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\user\echelon folder on your computer (or other location where the **Welcome.htm** file in your working copy of the SmartServer embedded image is stored).
2. Open the **Welcome.htm** file with your text editor.
3. Translate the highlighted lines of code.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2 <html>
3 <head>
4 <meta http-equiv="content-type" content="text/html; charset=utf-8"
5 <title>i.LON SmartServer - Bienvenido</title>
6 <link href="/styles/echelon/global.css" type="text/css" rel="stylesheet"
7 <script type="text/javascript" src="/scripts/echelon/EchelonInit.js"></script>
8 <style>
9 <!--
10 .welcome { color: #113a68; font-size: 24px; font-family: "Trebuchet MS", Verdana, Arial, sans-serif; font-sty
11 ==>
12 </style>
13 </head>
14 <body bgcolor="#ffffff" leftmargin="0" topmargin="0"
15 <table width="100%" border="0" cellspacing="0" cellpadding="0" height="100%"
16 <tr>
17 <td colspan="2" valign="top" width="100%"
18 <table width="100%" border="0" cellspacing="0" cellpadding="0" height="100%"
19 <tr>
20 <td colspan="2" height="501">
21 <td colspan="2" height="501">
22 <table width="100%" border="0" cellspacing="0" cellpadding="0">
23 <tr valign="top" height="15">
24 <td colspan="2" height="500" background=".../images/building.gif"
25 <td colspan="2" height="500" background=".../images/building.gif"
26 <td colspan="2" height="500" background=".../images/building.gif"
27 <td colspan="2" height="500" background=".../images/building.gif"
28 <td colspan="2" height="500" background=".../images/building.gif"
29 <td colspan="2" height="500" background=".../images/building.gif"
30 <td colspan="2" height="500" background=".../images/building.gif"
31 <td colspan="2" height="500" background=".../images/building.gif"
32 <td colspan="2" height="500" background=".../images/building.gif"
33 <td colspan="2" height="500" background=".../images/building.gif"
34 <td colspan="2" height="500" background=".../images/building.gif"
35 <td colspan="2" height="500" background=".../images/building.gif"
36 <td colspan="2" height="500" background=".../images/building.gif"

```

- Line 8 corresponds to the “i.LON SmartServer 2.0 – Welcome” title at the top of the SmartServer Welcome Web page.



- Lines 32–34 correspond to the “i.LON SmartServer 2.0 – Welcome” text at the bottom of the SmartServer Welcome Web page.




- Save the **Welcome.htm** file.
- Copy the **Welcome.htm** file to the SmartServer. To do this, follow these steps:
  - Browse to the LonWorks\iLon100\images\iLon100 4.0<language>\web\user\Echelon folder on your computer (or other location where the **Welcome.htm** file in your working copy of the SmartServer embedded image is stored).
  - Use FTP to access the root/web/user/echelon/<language[\_REGION] [\_variant]> folder on the SmartServer flash disk.
  - Copy the **Welcome.htm** file to the root/web/user/echelon/<language[\_REGION] [\_variant]> folder on the SmartServer flash disk.

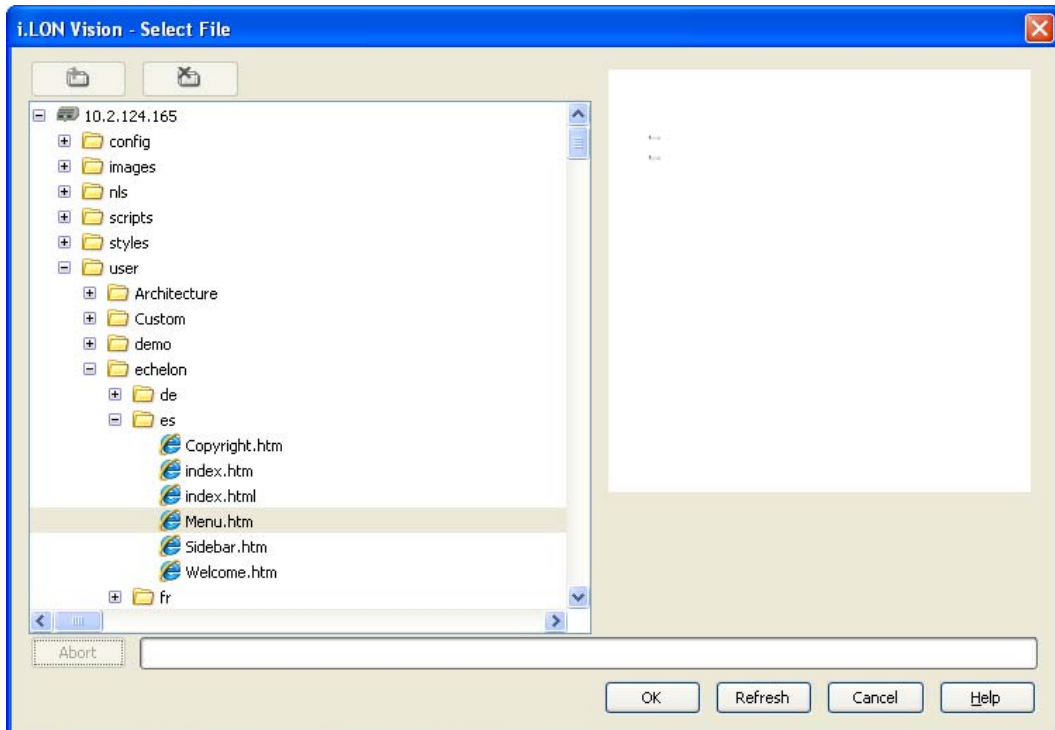
## Translating the Menu.htm File

You can translate the **Welcome.htm** and **Menu.htm** files in the `web/user/echelon/<language[_REGION] [_variant]>` folder in your working copy of the SmartServer embedded image. You can do the translation with i.LON Vision 2.0, or you can do it with a text editor.

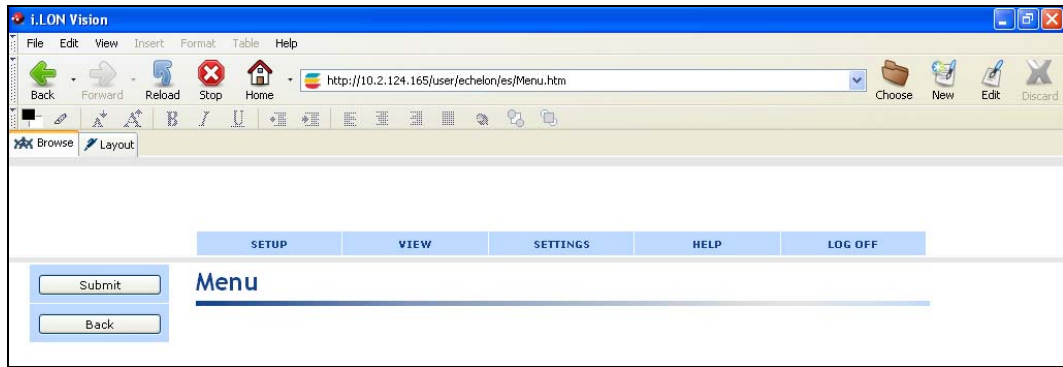
### Translating the Menu.htm File with i.LON Vision 2.0


You can translate the **Menu.htm** file using i.LON Vision 2.0. To do this, follow these steps:

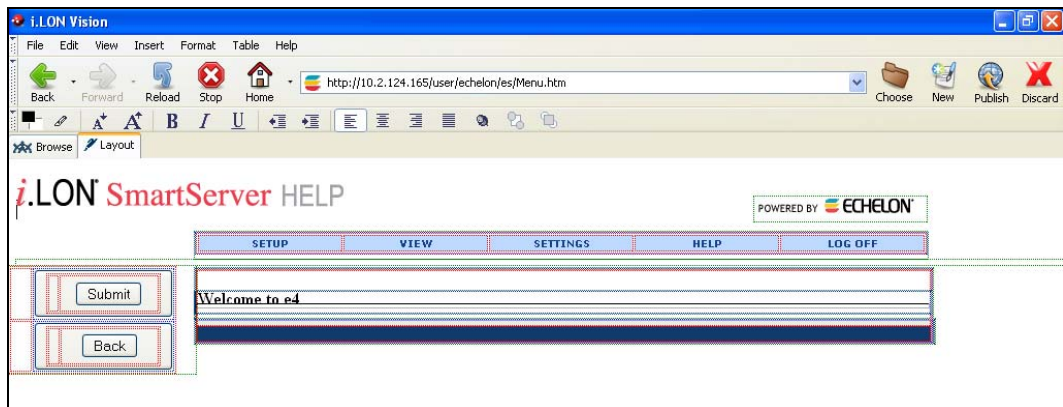
1. Verify that you copied your `web/user/echelon/<language[_REGION] [_variant]>` folder to your SmartServer following the steps described in *Translating the Welcome.htm File with i.LON Vision 2.0*.
2. Start i.LON Vision 2.0. To do this, click **Start**, point to **Programs**, point to Echelon **i.LON Vision 2.0 SmartServer 2.0**, and then click **i.LON Vision 2.0 SmartServer 2.0**. i.LON Vision 2.0 opens.
3. Connect i.LON Vision 2.0 to your SmartServer. To do this, click **Manage Connections** in the Sites pane on the left side, or click **File** and then click **Site Manager**. The **Site Manager** dialog opens. Click **New Site**, the **Edit Site** dialog opens. Enter your SmartServer's information, and then click **OK** twice. A link with the IP address of your SmartServer is added to the Sites pane.
4. Click the Choose button () on the Editor toolbar to open the **Select File** dialog opens. Select the `web/user/echelon/<language[_REGION] [_variant]>/Menu.htm` Web page, and then click **OK**.



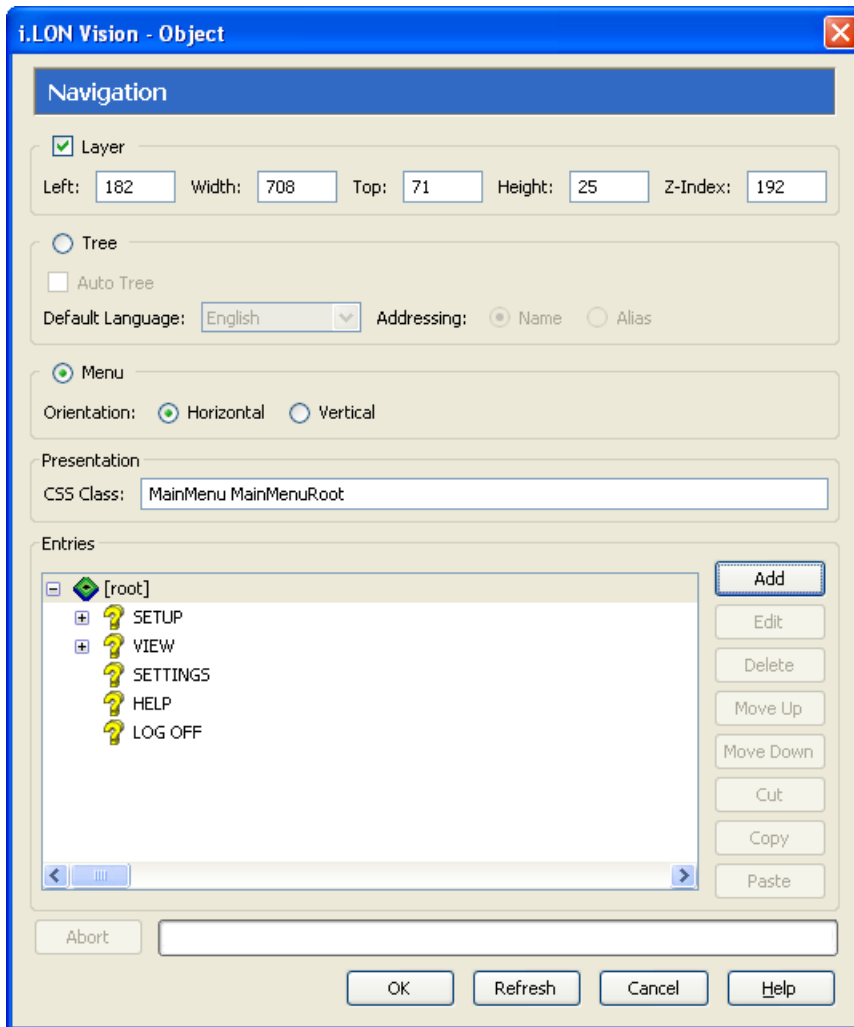
5. The English version of the **Menu.htm** Web page opens.



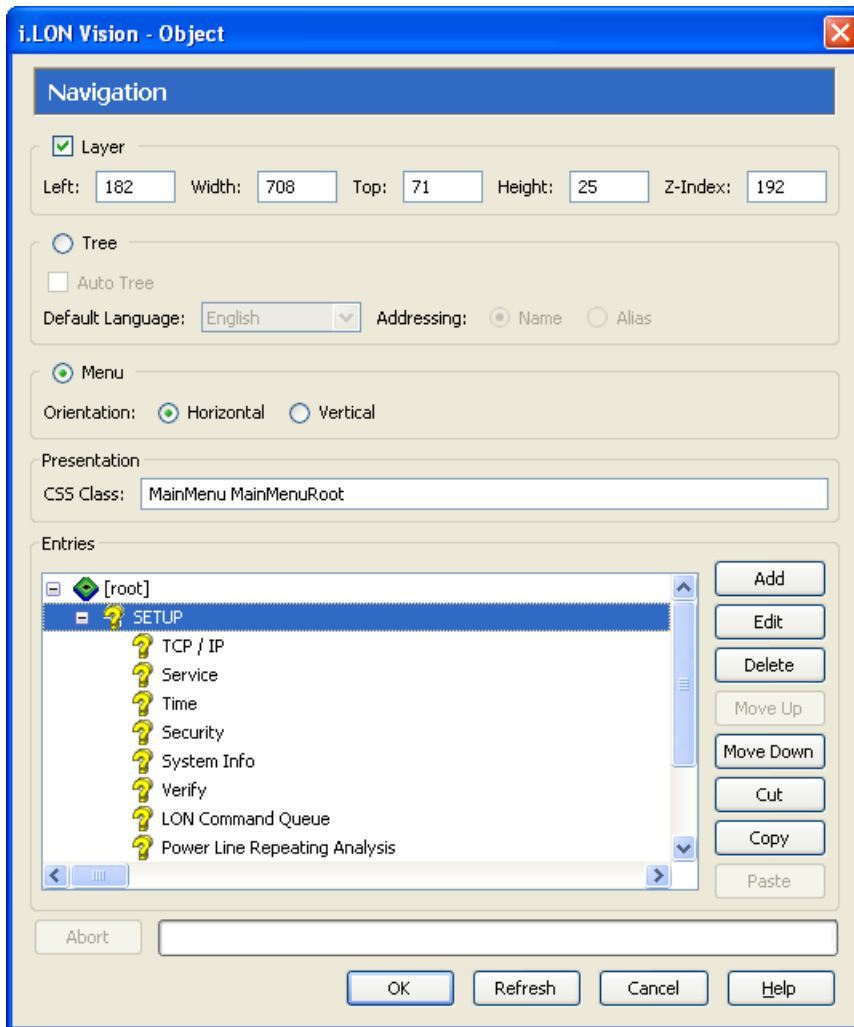
6. Click **Edit** on the Editor toolbar () or click the **Layout** tab.



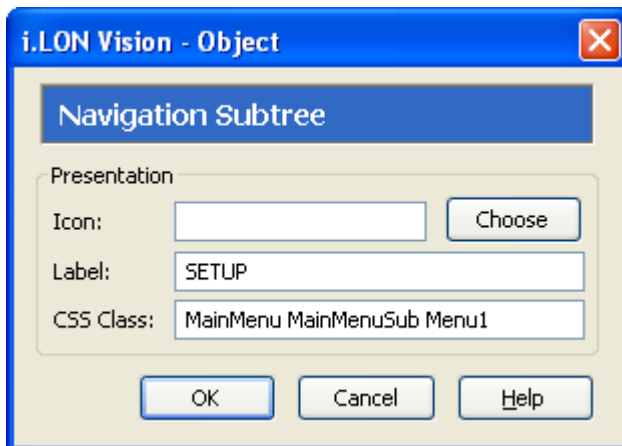
7. Double-click the Menu bar and the top of the Web page, or right-click the Menu bar and click **Object Properties** on the shortcut menu. The **Navigation Edit** dialog opens.



8. Expand **SETUP** to display its submenu items.



9. Click **Edit**. The **Navigation Subtree** dialog opens.



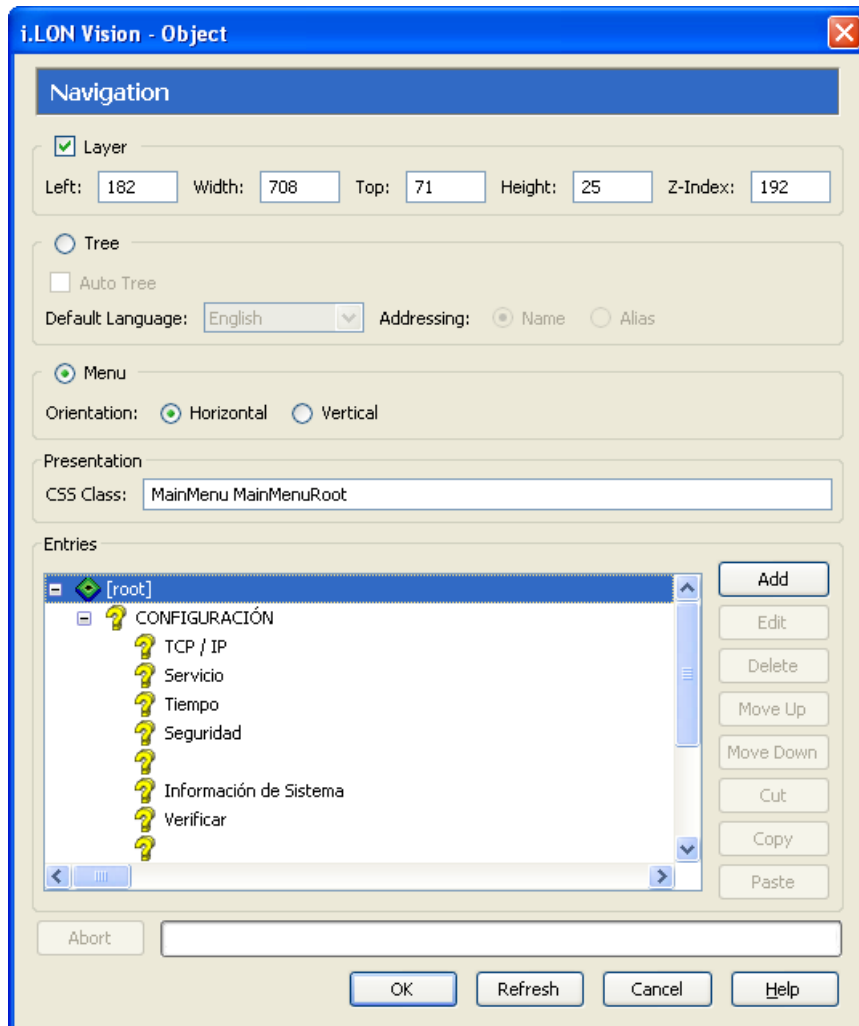
10. In the **Label** box, translate **SETUP** to your localized language and then click **OK**.

11. One-by-one, click the items listed under **SETUP**, click **Edit**, enter the translation in the **Label** box of the **Navigation Link** dialog, and then click **OK**.

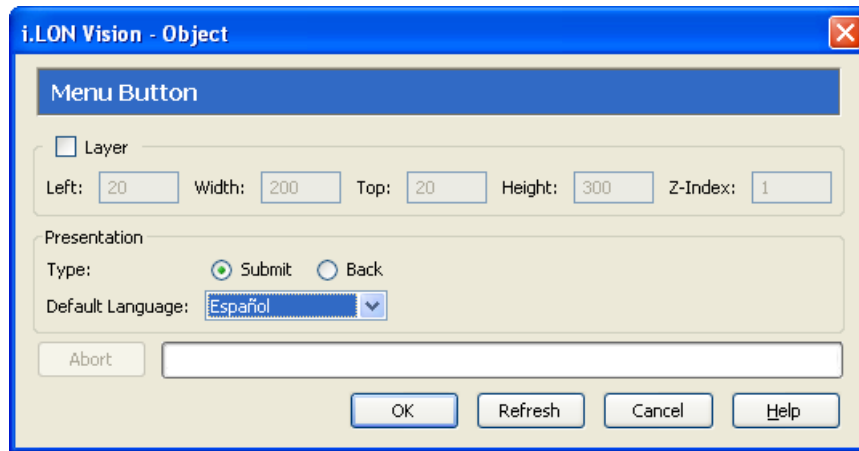
12. Repeat steps 9-12 to translate the remaining subtree and link menu items.



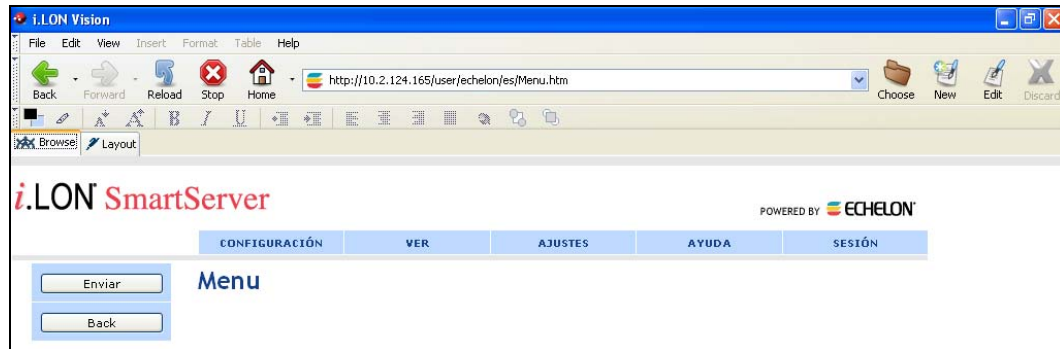
13. When you have finished translating the menus and menu items to your localized language, click **OK**.



14. Change the language of the **Submit** and **Back** buttons to your localized language. To do this, follow these steps:
- Double-click the **Submit** button, or right-click the **Submit** button and click **Object Properties** on the shortcut menu. The **Menu Button** dialog opens
  - Select your localized language from the **Default Language** box and then click **OK**. This sets your localized language as the default for the **Submit** button.



- c. Repeat steps a–b to change the language of the **Back** button to your localized language.
15. Click **Publish**. The **Menu.htm** Web page appears in your localized language.



### Translating the Menu.htm File with a Text Editor

You can translate the **Menu.htm** file using a text editor such as Notepad, WordPad, or TextPad. This entails translating the menu and menu items and updating the language settings in the file. Updating the language settings enables the SmartServer to display the “submit” and “back” buttons in the menu frame in your localized language.

To translate the **Menu.htm** file, follow these steps:

1. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\user\echelon folder on your computer (or other location where the **Menu.htm** file in your working copy of the SmartServer embedded image is stored).
2. Open the **Menu.htm** file with your text editor.
3. Translate the highlighted lines of code.

```

28 <div id="menuLayer" style="left: 182px;top: 71px;width: 708px;height: 25px;z-index: 192; position: absolute; ove
29 <div elon_type="menu" elon_orientation="horizontal" elon_event="static" elon_target="appFrame" elon_class="M
30 <div elon_arg_type="array" elon_arg_name="entry_arr">
31 <div elon_arg_type="object">
32 v elon_arg_name="label" elon_arg_val="CONFIGURACION"</div>
33 v elon_arg_name="class" elon_arg_val="MainMenu MainMenuSub Menu1"</div>
34 v elon_arg_name="shift_y" elon_arg_val="4"</div>
35 v elon_arg_type="array" elon_arg_name="entry_arr">
36
37
38 <div elon_arg_type="object" elon_label="TCP / IP">
39 <div elon_arg_type="object" elon_arg_name="vars_obj" elon_cm_type="localIlonIp" elon_cm_index="0"</div>
40 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/CM_LocalIlonIp_Cfg.htm" target="appFrame"></a>
41 </div>
42 <div elon_arg_type="object" elon_label="Servicio" elon_modal="false">
43 <div elon_arg_type="object" elon_arg_name="vars_obj" elon_cm_type="localIlonService" elon_cm_index="0"</div>
44 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/CM_LocalIlonService_Cfg.htm" target="appFrame"></a>
45 </div>
46 <div elon_arg_type="object" elon_label="Tiempo" elon_modal="false">
47 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/SetupTime.htm" target="appFrame"></a>
48 </div>
49 <div elon_arg_type="object" elon_label="Seguridad" elon_modal="false">
50 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/SetupSecurity.htm" target="appFrame"></a>
51 </div>
52 <div elon_arg_type="object" elon_line="true"</div>
53 <div elon_arg_type="object" elon_label="Información de Sistema" elon_modal="false">
54 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/SystemInfo.htm" target="appFrame"></a>
55 </div>
56 <div elon_arg_type="object" elon_label="Verificar" elon_modal="false">
57 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/SetupVerify.htm" target="appFrame"></a>
58 </div>
59 <div elon_arg_type="object" elon_line="true"</div>
60 <div elon_arg_type="object" elon_label="LON Mando Cola" elon_modal="false">
61 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/LonCommandQueue.htm" target="appFrame"></a>
62 </div>
63 <div elon_arg_type="object" elon_label="Línea Eléctrica de Repetir el Análisis" elon_modal="false">
64 <a elon_arg_tag elon_arg_name="url" href="/user/echelon/PowerlineRepeatingAnalysis.htm" target="appFrame"></a>
65 </div>
66 </div>
67 </div>
68 <div elon_arg_type="object" elon_label="VER" elon_class="MainMenu MainMenuSub Menu1" elon_shift_y="4">
69 <div elon_arg_type="array" elon_arg_name="entry_arr">
70 <div elon_arg_type="object" elon_label="Resumen de Alarma">
71 <a elon_arg_tag elon_arg_name="url" href="8000010128000000[4].UFPTAlarmaNotifier_Summary.htm" target="appFrame"></a>
72 </div>
73 <div elon_arg_type="object" elon_label="Historia de Alarma">
74 <a elon_arg_tag elon_arg_name="url" href="8000010128000000[4].UFPTAlarmaNotifier_History.htm" target="appFrame"></a>
75 </div>
76 <div elon_arg_type="object" elon_line="true"</div>
77 <div elon_arg_type="object" elon_label="Datos del Registro">
78 <a elon_arg_tag elon_arg_name="url" href="8000010128000000[4].UFPTdataLogger_View.htm" target="appFrame"></a>
79 </div>
80 <div elon_arg_type="object" elon_line="true"</div>
81 <div elon_arg_type="object" elon_label="Puntos de Datos">
82 <a elon_arg_tag elon_arg_name="url" href="ViewDataPoints.htm" target="appFrame"></a>
83 </div>
84 </div>
85 </div>
86 <div elon_arg_type="object" elon_no_scrolling="true" elon_label="AJUSTES" elon_shift_y="4">
87 <a elon_arg_tag elon_arg_name="url" href="globalsettings.htm" target="popup"></a>
88 </div>
89 <div elon_arg_type="object" elon_label="AYUDA">
90 <a elon_arg_tag elon_arg_name="url" href="javascript:EchelonTop.Echelon.Echelon.getInstance().showHelp()" target="popup"></a>
91 </div>
92 <div elon_arg_type="object" elon_label="SESIÓN">
93 <a elon_arg_tag elon_arg_name="url" href="shared/Logout.htm" target="popup"></a>
94 </div>
95 </div>
96 </div>
97 <div id="ilon2" style="visibility: visible; display: block; position: absolute; z-index: 190; top: 210px; left: 5

```

- Lines 32–44 in this example correspond to the “Setup” menu and its menu items.

CONFIGURACIÓN
TCP / IP
Servicio
Tiempo
Seguridad
Sistema de Información
Verificar
LON Mando Cola
Línea Eléctrica de Repetir el Análisis

- Lines 62–82 in this example correspond to the “View” menu and its menu items.

VER
Resumen de Alarma
Historia de Alarma
Datos del Registro
Puntos de Datos

- Lines 85–93 in this example correspond to the “Settings”, “Help”, and “Log Off” menus. Note that the translation of the SmartServer online help files is not supported.



4. Change the language for the **Submit** and **Back** buttons to “`elon_lang_<language[_REGION] [_variant]>`” (lines 121 and 127 in the following example).

```

Menu.htm*
91         <div elon_arg_type="object" elon_label="SESIÓN">
92             <a elon_arg_tag elon_arg_name="url" href="../../shared/Logout.htm" target="popup"></a>
93         </div>
94     </div>
95 </div>
96 </div>
97 <div id="ilon2" style="visibility: visible; display: block; position: absolute; z-index: 190; top: 21px; left: 5
98     
99 </div>
100 <div id="eLonPower" style="visibility: visible; display: block; position: absolute; z-index: 190; top: 36px; left:
101     
102 </div>
103
104 <div id="titlebar" style="position: absolute; z-index: 190; top: 106px; left: 182px; width: 711px; height: 25px"
105 <div elon_type="titlebar" elon_version_number="4.0"></div>
106 </div>
107
108 <div id="progress" style="position: absolute; z-index: 190; top: 131px; left: 182px; width: 711px;">
109     
111
112 <div id="messagebar" style="left: 182px; top: 155px; width: 711px; height: 23px; position: absolute; ">
113     <div elon_type="messagebar" elon_version_number="4.0" elon_lang="{elon_language}"></div>
114 </div>
115
116 <div id="buttonLayer" style="left: 0px; top: 103px; width: 182px; height: 77px; z-index: 190; position: absolute;">
117     <table id="buttonTable" border="0" cellspacing="1" cellpadding="0" height="100%">
118         <tr height="35">
119             <td class="Empty" width="19"></td>
120             <td id="submitCell" class="Background" align="center" width="135">
121                 <button class="ControlButton" elon_type="submitbutton" elon_lang="{elon_language+es}" elon_versi
122             </td>
123         </tr>
124         <tr height="35">
125             <td class="Empty" width="19"></td>
126             <td id="backCell" class="Background" align="center" width="135">
127                 <button class="ControlButton" elon_lang="{elon_language+es}" elon_type="backbutton"></button>
128             </td>
129         </tr>
130     </table>
131 </div>
132 </body>

```

5. Save the **Menu.htm** file.
6. Copy the **Menu.htm** file to the SmartServer. To do this, follow these steps:
  - a. Browse to the LonWorks\iLon100\images\iLon100 4.0<language>\web\user\Echelon folder on your computer (or other location where the **Menu.htm** file in your working copy of the SmartServer embedded image is stored).
  - b. Use FTP to access the root/web/user/echelon/<language[\_REGION] [\_variant]> folder on the SmartServer flash disk.
  - c. Copy the **Menu.htm** file to the root/web/user/echelon/<language[\_REGION] [\_variant]> folder on the SmartServer flash disk.


### Translating the Sidebar.htm File

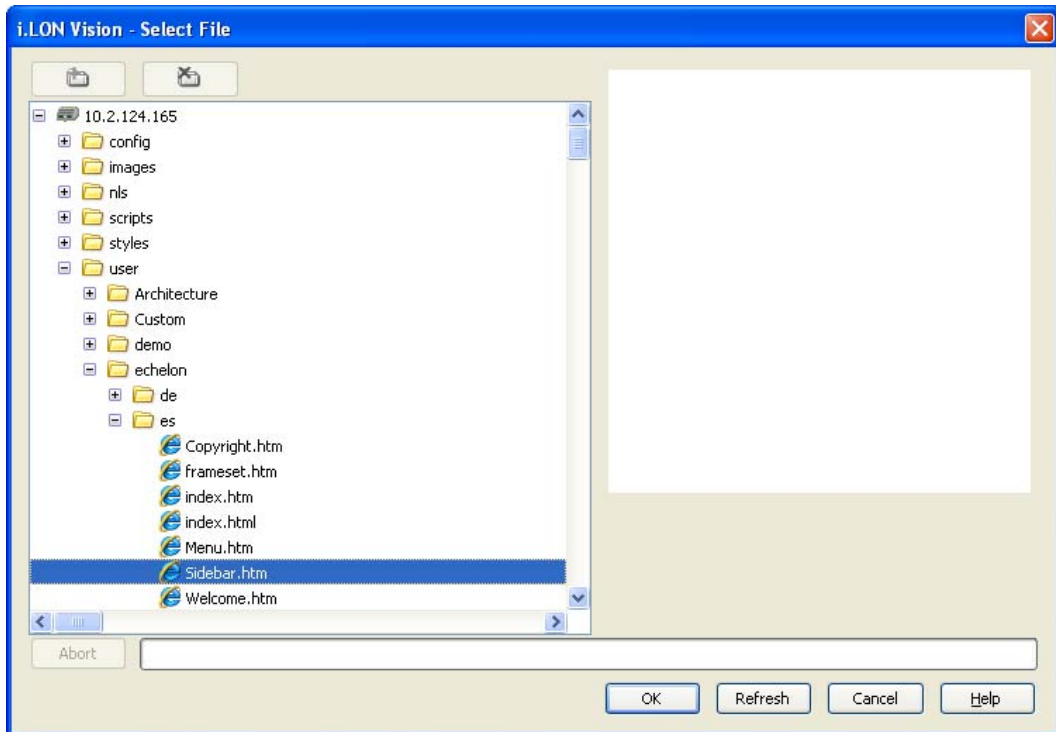
You can update the language settings of the **Sidebar.htm** files in the web/user/echelon/<language[\_REGION] [\_variant]> folder within your working copy of the SmartServer embedded image with i.LON Vision 2.0, or with a text editor. This enables the SmartServer to display the objects in the sidebar frame of the SmartServer Web interface in your localized language. The objects in the sidebar frame consist of the **General** and **Driver** mode buttons, the message box, and the objects in the navigation pane.

### Translating the Sidebar.htm File with i.LON Vision 2.0

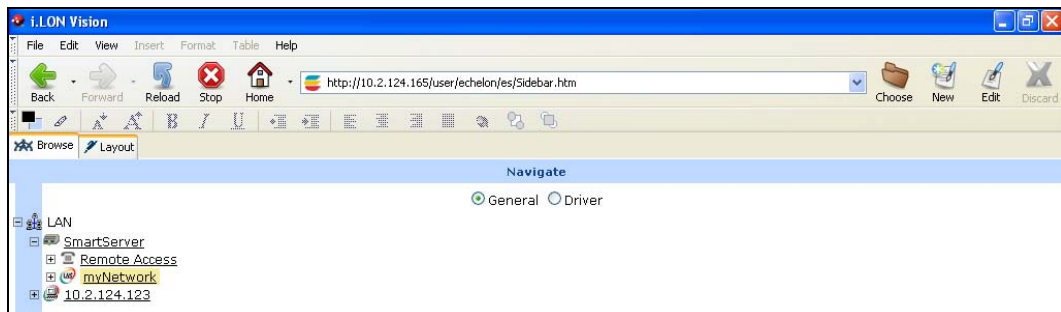
You can update the language settings for the **Sidebar.htm** file using i.LON Vision 2.0.


1. Verify that you copied your web/user/echelon/<language[\_REGION] [\_variant]> folder to your SmartServer following the steps described in *Translating the Welcome.htm File with i.LON Vision 2.0*.

2. Connect *i.LON Vision 2.0* to your SmartServer. To do this, click **Manage Connections** in the Sites pane on the left side, or click **File** and then click **Site Manager**. The **Site Manager** dialog opens. Click **New Site**, the **Edit Site** dialog opens. Enter your SmartServer's information, and then click **OK** twice. A link with the IP address of your SmartServer is added to the Sites pane.
3. Click the Choose button () on the Editor toolbar to open the **Select File** dialog opens. Select the web/user/echelon/<language[\_REGION] [\_variant]>/Sidebar.htm Web page, and then click **OK**.



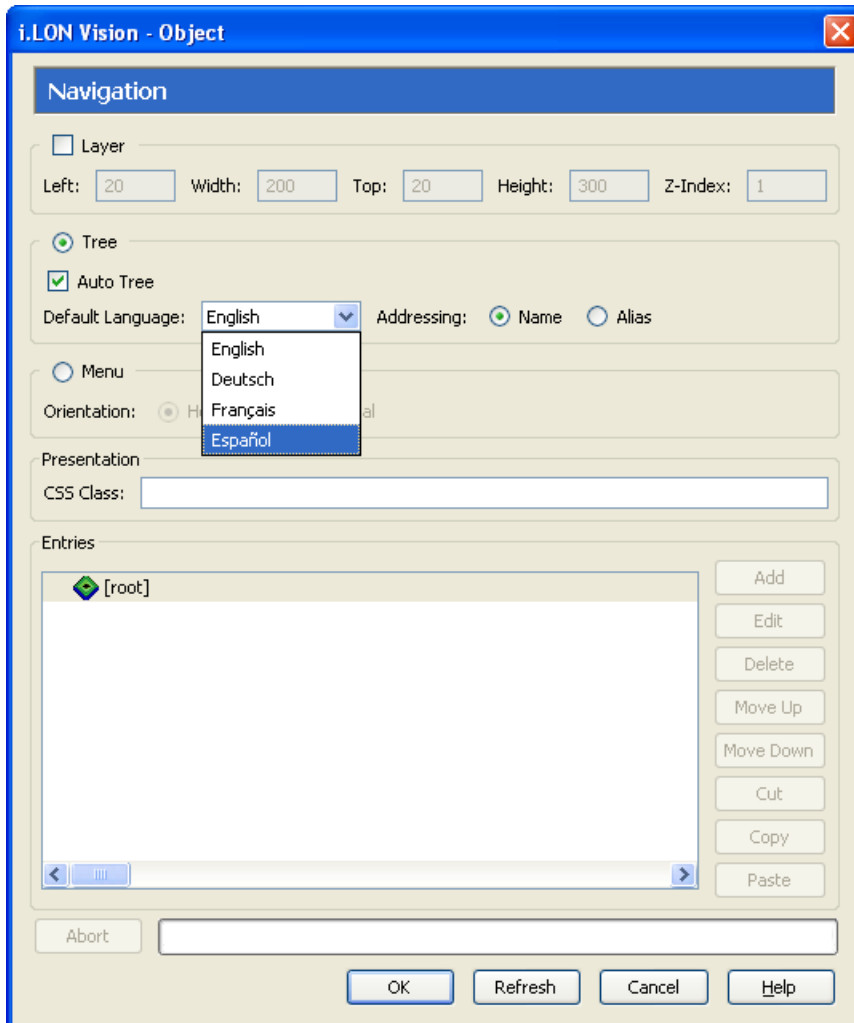
4. The English version of the **Sidebar.htm** Web page opens.



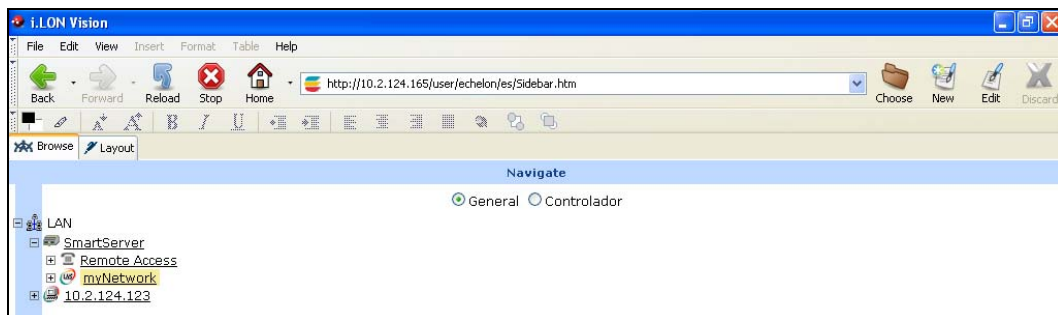
5. Click **Edit** on the Editor toolbar () or click the **Layout** tab. An auto tree navigation icon appears on the Web page.



- Double-click the Menu bar and the top of the Web page, or right-click the Menu bar and click **Object Properties** on the shortcut menu. The **Navigation Edit** dialog opens.
- Select your localized language from the **Default Language** box and then click **OK**. This sets your localized language as the default for the **Sidebar.htm** file.



- Click **Publish**. The **Sidebar.htm** Web page appears in your localized language.



### Translating the Sidebar.htm File with a Text Editor

You can update the language settings for the **Sidebar.htm** file using a text editor such as Notepad, WordPad, or TextPad. To do this, follow these steps:

1. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\user\Echelon folder on your computer (or other location where the **Sidebar.htm** file in your working copy of the SmartServer embedded image is stored).
2. Open the **Sidebar.htm** file with your text editor.
3. Change the one “elon\_lang\_de” setting to “elon\_lang\_<language[\_REGION] [\_variant]>” (line 12 in the following example).

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
6     <title>Sidebar</title>
7     <script type="text/javascript" src="/scripts/echelon/EchelonInit.js"></script>
8     <link href="/styles/echelon/Global.css" type="text/css" rel="stylesheet">
9   </head>
10  <body>
11
12  <div elon_type="navtree" elon_version_number="4.0" elon_lang="{elon_language=es}" elon_auto_tree="true" elon_system_d
13 </div>
14

```

4. Save the **Sidebar.htm** file.
5. Copy the **Sidebar.htm** file to the SmartServer. To do this, follow these steps:
  - a. Browse to the LonWorks\iLon100\images\iLon100 4.0<x> <Language>\web\user\Echelon folder on your computer (or other location where the **Sidebar.htm** file in your working copy of the SmartServer embedded image is stored).
  - b. Use FTP to access the root/web/user/echelon/<language[\_REGION] [\_variant]> folder on the SmartServer flash disk.
  - c. Copy the **Sidebar.htm** file to the root/web/user/echelon/<language[\_REGION] [\_variant]> folder on the SmartServer flash disk.

### Viewing the Localized SmartServer Web Interface

After you have copied the **index.htm** file and the <language[\_REGION] [\_variant]> folder to the SmartServer, you can view your localized version of the SmartServer Web interface. To do this follow, these steps:

1. Open your *i.LON* SmartServer 2.0’s home page. If the *i.LON* SmartServer 2.0 Web pages are already open, close your browser.
2. In the **Configuration & Service** box near the top of the home page, select your localized language.



3. Click **Login**. The localized **i.LON SmartServer 2.0 - Welcome** Web page opens.



4. The menus, the objects in the sidebar frame (left frame), and text in the **i.LON SmartServer 2.0 - Welcome** Web page (application frame to the right) should appear in your localized language.
5. Click the menus to view the translated menu items. Expand the tree in the sidebar frame and click the objects in the tree to see their translated Configuration and Driver property Web pages. Click on various embedded applications in **General** mode to view their translated versions.

**Tip:** If objects do not appear in their localized language, you may need to delete the temporary internet files from your computer.



# Appendix A

## FPM Programmer's Reference

This appendix details the files, routines and methods you will use to create and program your FPMs.

---

## Overview

This chapter provides details you will need when programming your module. It includes the following sections:

- *Template Files*. This section describes the template files you will use to create your module.
- *Routines*. This section describes the four main routines you will need to implement within your custom module: `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()`.
- *Methods*. This section describes the data point, timer, RS-232 interface, RS-485 interface, and file access methods you can call from the four main routines.

---

## Template Files

When you create an FPM, the following template files are generated for your module.

- **.cpp** file. This C++ source file contains the `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()` routines that specify the behavior of your FPM.
- **.h** file. This C header template file contains all the routine and method definitions for your FPMs.
- **\_Utils.cpp** file. This C++ source file contains all the helper routines called by the `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()` routines.

---

## Routines

The behavior of an FPM is defined by the `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()` routines that are called from the **.cpp** file. The following table displays when these routines are executed and the type of functions performed in each of these routines for an FPM application and an FPM driver.

Routine	When Routine is Executed	Functions to be Performed in FPM	
		FPM Application	FPM Driver
<code>Initialize()</code>	FPM is started or enabled	<ol style="list-style-type: none"><li>1. Set initial data point values.</li><li>2. Start timers.</li></ol>	<ol style="list-style-type: none"><li>1. Open RS-232 or RS-485 interface.</li><li>2. Start timers.</li><li>3. Write data point properties.</li></ol>
<code>Work()</code>	Data point value changes	<ol style="list-style-type: none"><li>1. Execute an algorithm.</li><li>2. Start and stop timers.</li><li>3. Read data point properties.</li></ol>	<ol style="list-style-type: none"><li>1. Initialize RS-232 or RS-485 interface.</li><li>2. Write to the RS-232 or RS-485 interface.</li></ol>
<code>OnTimer()</code>	Timer expires	<ol style="list-style-type: none"><li>1. Perform routine tasks such as reading data point status.</li><li>2. Read other data point properties.</li><li>3. Start and stop timers.</li></ol>	<ol style="list-style-type: none"><li>1. Initialize RS-232 or RS-485 interface.</li><li>2. Read and write to RS-232 or RS-485 interface.</li><li>3. Write values to data points.</li></ol>
<code>Shutdown()</code>	FPM is stopped or disabled	<ol style="list-style-type: none"><li>1. Stop timers.</li></ol>	<ol style="list-style-type: none"><li>1. Stop timers.</li><li>2. Close RS-232 or RS-485 connection.</li></ol>

---

## Initialize()

The `Initialize()` routine in the `.cpp` file is called when your FPM application or driver starts or is enabled. For an FPM application, you can use the `Initialize()` routine to write initial data point values, and start timers. For an FPM driver, you can use the `Initialize()` routine to open RS-232 or RS-485 connections, start timers, and write data point properties.

- You can start timers using the `Start()` method of the `CFPM_Timer` class or the user-defined `START_TIMER()` macro. The `Start()` method calls back the `OnTimer()` routine, which handles timers expiration events. The `START_TIMER()` macro calls back a custom timer handler that you must create. See *Timer Methods* for more information about using the `Start()` method and the `START_TIMER()` macro.
- You can open the RS-232 and RS-485 interfaces on the SmartServer using the `rs232_open()` and `rs485_open()` methods. For more information on these methods, see *RS-232 Interface Methods* and *RS-485 Interface Methods* later in this chapter.

### FPM Application Example

The following example demonstrates code you could use in the `Initialize()` routine of an FPM Application. In this example, an initial value is written to a data point and two timers are started.

```
DECLARE(_0000000000000000_0_::SNVT_temp_f, nviSetPoint,
INPUT_DP)

CFPM_Timer m_oTimer1; //declared in header file
CFPM_Timer m_oTimer2; //declared in header file

void CUFPT_FPM_Application::Initialize()
{
    nviSetPoint = 20.28;

    m_oTimer1.Start(FPM_TF_REPEAT, 2000);
    m_oTimer2.Start(FPM_TF_ONETIME, 0);
}
```

### FPM Driver Example

The following example demonstrates code you could use in the `Initialize()` routine of an FPM Driver. In this example, the FPM connects to an RS-232 interface and then starts one timer.

```
int _rs232_fd = -1;
CFPM_Timer m_oTimer3; //declared in header file

void CUFPT_FPM_Driver::Initialize()
{
    _rs232_fd = rs232_open(9600);
    m_oTimer3.Start(FPM_TF_REPEAT, 1200);
}
```

---

## Work()

The `Work()` routine in the `.cpp` file is called when the value of a data point declared in an FPM application or driver changes. The `Work()` routine establishes functionality between the data points defined in the FPM to the data points on the SmartServer. When configuring the `Work()` routine, you determine which data point was updated using the `Changed()` method. See *Methods* for more information on using this method.

For an FPM application, you can also use the `Work()` routine to start and stop timers, and you can use it to read data point properties.

For an FPM driver, if a data point value has changed and you want to write data to the RS-232 interface as a result, you must first initialize communication between your FPM and the devices connected to the RS-232 and RS-485 interfaces. You can initialize communication with the interfaces using the `rs232_ioctl()` and `rs485_ioctl()` methods.

### *FPM Application Example*

The following example demonstrates code you could use in the `Work()` routine of an FPM application. In this example, the `Changed()` method evaluates whether the values of two data points has changed and performs some algorithm if at least one of the values has changed.

```
SNVT_count in1;
SNVT_count in2;

void CUFPT_FPM_Application::Work() {

    if (Changed(in1) || Changed(in2))
    {
        // perform some algorithm when value of in1 or in2 changes
    }
}
```

### *FPM Driver Example*

The following example demonstrates code you could use in the `Work()` routine of an FPM driver. In this example, the `Changed()` method evaluates whether the value of a data point has changed and initializes and writes to the RS-232 interface if it has.

```
SNVT_str_asc Line1;

void CUFPT_FPM_Driver::Work()
{

    if (Changed(Line1))
    {
        //Initialize RS-232 interface
        int nBytesToRead;
        rs232_ioctl(RS232_fd, FIONREAD, (int) &nBytesToRead);

        // Write to RS-232 interface
        rs232_write(RS232_fd,
                   (Byte *)Line1->ascii,
                   strlen((char*)Line1->ascii));
    }
}
```

---

## *OnTimer()*

The `OnTimer()` routine (or your custom timer handler) in the `.cpp` file handles timer expiration events. You use this routine in conjunction with the `Start()` methods and the `START_TIMER()` macros called in the `Initialize()` routine. When configuring the `OnTimer()` routine, you can determine which timer expired using the `m_oTimer.Expired()` method. See *Timer Methods* for more information on using these timer methods.

For an FPM application, you can also use the `OnTimer()` routine to start and stop timers, and you can use it to read and write to data point properties. For an FPM driver, you can use the `OnTimer()` routine to read and write to the RS-232 and RS-485 interfaces.

### *FPM Application*

The following code demonstrates code you could use in the `OnTimer()` routine in an FPM application. In this example, the `m_oTimer.Expired()` method evaluates which of two timers has expired and performs some tasks upon their expiration. If `m_oTimer1` expires, the `OnTimer()` routine checks whether the status of a data point is `AL_ALM_CONDITION`. If `m_oTimer2` expires, the `OnTimer()` routine re-starts the timer.

```
CFPM_Timer m_oTimer1; //declared in header file
CFPM_Timer m_oTimer2; //declared in header file

m_oTimer1.Start(FPM_TF_REPEAT, 2000);
m_oTimer2.Start(FPM_TF_ONETIME, 0);

void CUFPT_FPM_Application::OnTimer()
{
    if (m_oTimer1.Expired())
    {
        //check status of a data point
        nviTemp_status =
        nviTemp.GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus);

        if (nviTemp_status == AL_ALM_CONDITION)
        {
            //perform algorithm if nviTemp status is AL_ALM_CONDITION
        }
    }

    if (m_oTimer2.Expired())
    {
        //restart a timer
        m_oTimer2.Start(FPM_TF_ONETIME, 1000);
    }
}
```

### *FPM Driver*

The following example demonstrates code you could use in the `OnTimer()` routine of an FPM driver. In this example, a custom timer handler method evaluates whether a timer started with the `START_TIMER` method has expired. If the timer has expired, the custom timer handler method reads and writes to the RS-232 interface.

```
CFPM_Timer m_oTimer3; //declared in header file
START_TIMER(m_oTimer3, FPM_TF_REPEAT, 10000, RS_232_Timer);

void CUFPT_FPM_Driver::RS_232_Timer()
{
    if (rs232_read(RS232_fd, Line1, 1) == 1)
    {
        rs232_write(RS232_fd, (Byte *)"F1", strlen("F1"));
    }
}
```

---

## Shutdown()

The `Shutdown()` routine in the `.cpp` file is called when your FPM stops or is disabled. You can use the `Shutdown()` routine to stop timers and close RS-232 and RS-485 connections in an FPM driver. In addition, you can use `Shutdown()` routine to free previously allocated memory and perform any required cleanup before shutting down the FPM.

- You can stop a timer using use the `Stop()` and `StopAllTimers()` methods of the `CFPM_Timer` class.
- You can end communication between your FPM and the devices connected to the RS-232 and RS-485 serial ports on the SmartServer using the `rs232_close()` and `rs485_close()` methods. For more information on these methods, see *RS-232 Interface Methods* and *RS-485 Interface Methods* later in this appendix.

### FPM Application Example

The following example demonstrates code you could use in the `Shutdown()` routine of an FPM Application. In this example, a timer is stopped.

```
void CUFPT_FPM_Application::Shutdown()
{
    m_oTimer1.Stop;
}
```

### FPM Driver Example

The following example demonstrates code you could use in the `Shutdown()` routine of an FPM driver. In this example, the FPM closes the connection to an RS-232 interface and then stops all running timers.

```
void CUFPT_FPM_Driver::Shutdown()
{
    rs232_close(_rs232_fd);
    StopAllTimers();
}
```

---

## Methods

This section describes the methods provided by Echelon that you can use when writing the `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()` routines in your FPM. These methods include data point methods, timer methods, RS-232 methods, and RS-485 methods.

---

## Variable Types

The variable types that you can use when calling the `Initialize()`, `Work()`, `OnTimer()`, and `Shutdown()` routines in your FPM are listed in the following table. These variable types are defined in the `types_base.h` file in the `LonWorks\iLON\Development\compiler\echelon\fpm\include-4.03` and `LonWorks\iLON\Development\compiler\echelon\fpm\include` folders on your computer.

Variable Type	Size
Byte (equivalent to unsigned char)	8 Bits
unsigned char	8 Bits
char	8 Bits
unsigned short	16 Bits
short	16 Bits

Variable Type	Size
unsigned long	32 Bits
long	32 Bits
unsigned int	32 Bits
int	32 Bits
float	32 Bits
double	64 Bits

## Internal FPM Data Point Methods

For the data points declared in an FPM application or FPM driver, you can use the `Changed()` method to determine if a data point value has been changed, and you can use the `NotifyOnAllUpdates()` method to modify the `Changed()` method so that it checks whether any data point property has been updated, including value, status, time of last update, and priority.

You can use the `PROPAGATE()` macro to write values to a structured data point.

For data points declared in an FPM application, you can use the `ResetPriority()` method to set and reset the priority used by the module to write values to a data point.

You can use these internal data point methods in the `Work()` and `OnTimer()` routines of an FPM application or driver.

Internal Data Point Method	FPM Scope			
	Initialize()	Work()	OnTimer()	Shutdown
Changed()	—	FPM Application FPM Driver	FPM Application FPM Driver	—
NotifyOnAllUpdates()*	—	—	—	—
Propagate()	—	FPM Application FPM Driver	FPM Application FPM Driver	—
ResetPriority()	—	FPM Application	FPM Application	—

\*Declared in constructor of FPM Application or FPM Driver

### Changed()

You can use the `Changed()` method in the `Work()` and `OnTimer()` routines of an FPM application or driver to determine whether the value of a data point has changed.

#### SYNTAX

```
bool Changed(const CVariable& rVar)
```

The `rVar` parameter specifies a data point declared in the FPM. If the value of the specified data point has changed, this method returns a `true` value; otherwise, it returns a `false` value.

#### EXAMPLE

The following example demonstrates how you can use the `Changed()` method to check whether the values of the data points in your FPM have changed.

```
void CUFPT_FPM_Application::Work()
{
    if (Changed(x) || Changed(y))
    {
        //insert code here
    }
}
```

```
{
```

**Note:** You can use the `NotifyOnAllUpdates()` method modify so that the `Changed()` method checks whether any data point property has changed, including value, status, time of last update, and priority. See *NotifyOnAllUpdates()* for more information.

### *NotifyOnAllUpdates()*

You can call the `NotifyOnAllUpdates()` method in the constructor of your FPM so that the `Changed()` method checks whether any data point property has changed, including value, status, time of last update, and priority. Note that if you do not call this method within the FPM constructor, the `Changed()` method only checks whether the data point value has changed.

#### **SYNTAX**

```
void NotifyOnAllUpdates(const vector<string> &rDpVarNames);
```

The `rDpVarNames` parameter specifies a string vector contain the names of data points declared in the FPM for which the `Changed()` method is to check for updates to any of their properties (value, status, time of last update, and priority).

#### **EXAMPLE**

The following example demonstrates how to use the `NotifyOnAllUpdates()` and the `Changed()` methods to check whether the properties of specific data points in your FPM application have been updated.

```
// FPM constructor
vector<string> oDpVarNames;
oDpVarNames.push_back("x");
oDpVarNames.push_back("y");
NotifyOnAllUpdates(oDpVarNames);

.....

void CUFPT_FPM_Application::Work()
{
    if (Changed(x) || Changed(y))
    {
        //insert code here
    }
}
```

### *Propagate()*

You can use the `PROPAGATE()` macro in the `Work()` and `OnTimer()` routines of an FPM application or driver in conjunction with the `->` operator (element selection through pointer) to update the value of a structured data point.

#### **SYNTAX**

```
void PROPAGATE(varName)
```

The `varName` parameter specifies a data point declared in the FPM to be updated.

#### **EXAMPLE**

The following example demonstrates how to write to structured data points using the `->` operator and the `PROPAGATE()` method.

```
nvoSwitch->value = 200;
nvoSwitch->state = 1;
PROPAGATE(nvoSwitch);
```



**Note:** You can also use temporary data point variables to write values to the fields within a structured data point. To do this, you declare a temporary data point, store the desired values in the various fields of the temporary data point, and then assign the declared data point a reference to the temporary data point variable. The following code demonstrates how to write to a structured data point using temporary data point variables

```
SNVT_switch tmp_switch;    //create temporary DP variable

tmp_switch.value = 200;    // set DP value in temp DP variable
tmp_switch.state = 1;     // set DP state in temp DP variable
nvoSwitch = tmp_switch;   // assign declared DP to temp DP,
                          // which triggers a data point write
```

### *Write()*

You can perform a `Write()` method in an FPM application to explicitly write values to the data points declared in an FPM. Typically, you do not need to use this method because updated values are automatically propagated to the SmartServer's internal data server after the `Work()` and `OnTimer()` routines are done. However, you can use this method to send multiple data point updates within a single routine.

#### **SYNTAX**

```
STATUS Write(CVariable& rVar);
```

The `rVar` parameter specifies a data point declared in the FPM.

The `STATUS` value returned by this method can either be `ERROR` or `OK`.

#### **EXAMPLE**

The following example demonstrates a `Write()` method that changes the value of a data point declared in an FPM application.

```
DECLARE(_0000000000000000_0_::SNVT_temp_f, nviSetPoint,
INPUT_DP)
...
nviSetPoint = 25.83;
STATUS st = Write(nviSetPoint);

if (st==ERROR) //if dp write failed
{
    printf("DP write failed");
}
```

**Notes:** Using this `Write()` method in an FPM application may significantly impact the performance of the SmartServer; therefore, it is recommended that you use this method sparingly.

### *ResetPriority()*

You can use the `ResetPriority()` method in the `Work()` and `OnTimer()` routines of an FPM application to enable lower priority applications to write values to a data point declared in the module.

#### **SYNTAX**

```
STATUS ResetPriority(cVariable& rVar,
                    unsigned short nPrioAuthority);
```

The `rVar` parameter specifies the name of a data point declared in the FPM application.

The `nPrioAuthority` parameter specifies the priority to be used to reset the priority assigned to the data point. The priority is a value between 0–255 (highest priority) and 255 (lowest priority) that determines whether an application can write values to a data point.

If the priority specified in the `nPrioAuthority` parameter is equal to or higher than the priority currently assigned to the data point, this method returns a STATUS value of OK and the priority used by the FPM application to write to the data point is reset to 255. All applications in which the data point is registered are notified that they can write values to the data point, and the application with highest priority is able to write values to the data point.

To include multiple instances of an FPM in the calculation of the data point priority, you must use the `SetDpProperty(UCPTpriority)` method to set the **cfgUCPTpriority** property of the data point accordingly.

If the priority level specified is lower than the priority currently assigned to the data point, this method returns a STATUS value of ERROR.

**EXAMPLE**

The following example demonstrates a `ResetPriority()` method that attempts to reset the priority of a data point declared in an FPM application using a priority of 200.

```
STATUS st = ResetPriority (nviSetPoint, 200);

if (st==ERROR) //if current DP priority >200
{
    printf("DP Reset Priority too Low");
}
```

## *FPM Application Data Point Property Methods*

For the data points declared in an FPM application, you can use specific data point property methods in the `Work()` and `OnTimer()` routines get their names, times of last update, and statuses, and to get and set their priorities. The following table displays the valid scope of the data point property methods in an FPM application.

Property Name	FPM Application Scope			
	Initialize()	Work()	OnTimer()	Shutdown()
FPM::Dp::cfgUCPTname	—	Read	Read	—
FPM::Dp::cfgUCPTAliasName	—	Read/Write	Read/Write	—
FPM::Dp::dataUCPTlastUpdate	—	Read	Read	—
FPM::Dp::dataUCPTstatus	—	Read	Read	—
FPM::Dp::dataUCPTpriority	—	Read/Write	Read/Write	—

**Notes:**

- You can also use these data point property methods to get the properties of the external data points on the SmartServer. External data points include those data points on the internal SmartServer device [**i.LON App (Internal)**] and the data points of the external devices connected to the SmartServer.
- Using the data point property methods extensively may significantly impact the performance of the SmartServer; therefore, it is recommended that you use these methods sparingly.

### *GetDpPropertyAsString(UCPTname)*

You can use the `GetDpPropertyAsString(name)` method in the `Work()` and `OnTimer()` routines of an FPM application to read the name of the data point.

**SYNTAX**

```
const char* GetDpPropertyAsString(FPM::Dp::cfgUCPTname)
```

This method returns the **UCPTname** configuration property of the data point in the following format: `<network><channel><device><functional block><data point>`. The **UCPTname** configuration property is an array unsigned ASCII characters.

#### EXAMPLE

```
const char* nviSetPoint_name;  
  
nviSetPoint_name =  
nviSetPoint.GetDpPropertyAsString(FPM::Dp::cfgUCPTname);  
  
printf("nviSetPoint name = '%s' \n", nviSetPoint_name);
```

**Note:** Using this data point property method extensively may significantly impact the performance of the SmartServer; therefore, it is recommended that you use it sparingly.

#### *GetDpPropertyAsString(UCPTAliasName)*

You can use the `GetDpPropertyAsString(UCPTAliasName)` method in the `Work()` and `OnTimer()` routines of an FPM application to read the alias name of the data point. This is useful for getting the alias names of the external data points on the SmartServer.

#### SYNTAX

```
const char* GetDpPropertyAsString(FPM::Dp::AliasName)
```

This method returns the **UCPTAliasName** configuration property of the data point. This configuration property is defined in the **Alias Name** property in the data point's **Configuration – Data Point** Web page on the SmartServer.

The alias name was the naming convention used for data points in the *e3* release of the i.LON software. The data points in the tree were organized by their alias names, which correspond to the locations of the data points.

- The data points on the i.LON App (Internal) device under the LON channel have default alias names that begin with the "NVL" prefix.
- The virtual data points on the i.LON System (Internal) device under the VirtCh channel have default alias names that begin with the "iLON System" prefix. In the *e3* release of the i.LON software, these data points were referred to as "NVVs".
- The data points of the external devices connected to the SmartServer do not have default alias names, and this property is initially disabled for these data points. In the *e3* release of the i.LON software, these data points were referred to as "NVEs".

#### EXAMPLE

```
FpmItemInfoColl_t::iterator itEnd = oRTCiLon.end();  
for(FpmItemInfoColl_t::iterator it = oRTCiLon.begin();  
    it != itEnd; ++it)  
{  
    CFpmItemInfo &v = (*it);  
    printf("UCPTAliasName: %s,  
          GetDpPropertyAsString(Dp::cfgUCPTAliasName));  
}
```

**Note:** Using this data point property method extensively may significantly impact the performance of the SmartServer; therefore, it is recommended that you use it sparingly.

#### *GetDpPropertyAsTimeSpec(UCPTlastUpdate)*

You can use the `GetDpPropertyAsTimeSpec(UCPTlastUpdate)` method in the `Work()` and `OnTimer()` routines of an FPM application to read the time at which the data point was last updated.

## SYNTAX

```
const timespec GetDpPropertyAsTimeSpec(FPM::Dp::dataUCPTlastupdate)
```

This method returns the **UCPTlastupdate** configuration property of the data point in the following format: YYYY-MM-DDTHH:MM:SSZ. The **UCPTlastupdate** configuration property is a timestamp in UTC (Coordinated Universal Time) indicating the last time the data point configuration was updated.

## EXAMPLE

```
timespec nviSetPoint_lastUpdateTime;  
  
nviSetPoint_lastUpdateTime =  
nviSetPoint.GetDpPropertyAsTimeSpec(FPM::Dp::dataUCPTlastUpdate);  
  
printf ("SetPoint last update = %d\n",  
        nviSetPoint_lastUpdateTime);
```

**Note:** Using this data point property method extensively may significantly impact the performance of the SmartServer; therefore, it is recommended that you use it sparingly.

## *GetDpPropertyAsPointStatus(UCPTstatus)*

You can use the `GetDpPropertyAsPointStatus(UCPTstatus)` method in the `Work()` and `OnTimer()` routines of an FPM application to read the current status of a data point.

## SYNTAX

```
FPM::Dp::PointStatus  
GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus)
```

This method returns the **UCPTstatus** configuration property of the data point. The **UCPTstatus** configuration property is an enumeration defined by `enum PointStatus` in the **FPM\_Variable.h** file. It indicates the current status of the data point. This property is updated in real time by the SmartServer's internal data server. The enumerated values of this property are as follows:

AL_INVALID	= -1
AL_NO_CONDITION	= 0
AL_ALM_CONDITION	= 1
AL_TOT_SVC_ALM_1	= 2
AL_TOT_SVC_ALM_2	= 3
AL_TOT_SVC_ALM_3	= 4
AL_LOW_LMT_CLR_1	= 5
AL_LOW_LMT_CLR_2	= 6
AL_HIGH_LMT_CLR_	= 7
AL_HIGH_LMT_CLR_2	= 8
AL_LOW_LMT_ALM_1	= 9
AL_LOW_LMT_ALM_2	= 10
AL_HIGH_LMT_ALM_1	= 11
AL_HIGH_LMT_ALM_2	= 12
AL_FIR_ALM	= 13
AL_FIR_PRE_ALM	= 14
AL_FIR_TRBL	= 15
AL_FIR_SUPV	= 16
AL_FIR_TEST_ALM	= 17
AL_FIR_TEST_PRE_ALM	= 18
AL_FIR_ENVCOMP_MAX	= 19

AL_FIR_MONITOR_COND	=20
AL_FIR_MAINT_ALERT	=21
AL_FATAL_ERROR	=30
AL_ERROR	=31
AL_WARNING	=32
AL_HEADER	=243
AL_FOOTER	=244
AL_DEBUG	=245
AL_INFO	=246
AL_SYSTEM_INFO	=250
AL_VALUE_INVALID	=251
AL_CONSTANT	=252
AL_OFFLINE	=253
AL_UNKNOWN	=254
AL_NUL	=255

### EXAMPLE

```
FPM::Dp::PointStatus nviSetPoint_status;

nviSetPoint_status =
nviSetPoint.GetDpPropertyAsPointStatus(FPM::Dp::dataUCPTstatus);
printf("nviSetPoint status = %d \n \n", nviSetPoint_status);
```

**Note:** Using this data point property method extensively may significantly impact the performance of the SmartServer; therefore, it is recommended that you use it sparingly.

### *GetDpPropertyAsInt(UCPTpriority)*

You can use the `GetDpPropertyAsInt(UCPTpriority)` method in the `Work()` and `OnTimer()` routines of an FPM application to read the current priority of a data point.

**Note:** The use of the `GetDpPropertyAsInt(UCPTpriority)` method is not recommended.

### SYNTAX

```
int GetDpPropertyAsInt(FPM::Dp::dataUCPTpriority)
```

The `FPM::Dp::dataUCPTlastupdate` parameter specifies the **UCPTpriority** configuration property of the data point. The **UCPTpriority** configuration property is a short that indicates the current priority level assigned to the data point, where 0 is the highest priority, and 255 is the lowest.

### EXAMPLE

```
int nviSetPoint_priority;

nviSetPoint_priority =
nviSetPoint.GetDpPropertyAsInt(FPM::Dp::dataUCPTpriority);

printf("nviSetPoint priority = %d \n", nviSetPoint_priority);
```

### *SetDpProperty(UCPTAliasName)*

You can use the `SetDpProperty(UCPTAliasName)` method in the `Work()` and `OnTimer()` routines of an FPM application to write an alias name to a data point. This method updates the **UCPTAliasName** configuration property of the data point.

### SYNTAX

```
void SetDpProperty(FPM::Dp::dataUCPTAliasName, const char* const
pszValue)
```

The `pszValue` parameter specifies an alias name to be assigned the data point. The alias name can be any string that describes the data point.

**EXAMPLE**

```
nviSetPoint.SetDpProperty(FPM::Dp::dataUCPTAliasName, "FPM DP
Setpoint");
```

*SetDpProperty (UCPTpriority)*

You can use the `SetDpProperty(UCPTpriority)` method in the `Work()` and `OnTimer()` routines of an FPM application to write a priority to a data point. This method updates the **UCPTpriority** configuration property of the data point, which indicates the current priority level assigned to the data point, where 0 is the highest priority, and 255 is the lowest.

**Note:** The use of the `SetDpProperty(FPM::Dp::dataUCPTpriority)` method is not recommended.

**SYNTAX**

```
void SetDpProperty(FPM::Dp::dataUCPTpriority, int nValue)
```

The `nValue` parameter specifies the priority to be assigned the data point.

**EXAMPLE**

```
nviSetPoint.SetDpPropertyAsInt(FPM::Dp::dataUCPTpriority, 200);
```

## ***FPM Driver Data Point Property Methods***

For the data points declared in an FPM driver, you can use specific property methods in the `Initialize()` routine to set their default values, persistent flags, poll rates, and unit strings. The following table displays the valid scope of the data point property methods in an FPM driver.

Property Name	FPM Driver Scope			
	Initialize()	Work()	OnTimer()	Shutdown
FPM::Dp::cfgUCPTdefOutput	Write	—	—	—
FPM::Dp::cfgUCPTpersist	Write	—	—	—
FPM::Dp::cfgUCPTpollRate	Write	—	—	—
FPM::Dp::cfgUCPTunit	Write	—	—	—

**Note:** Using the data point property methods extensively may significantly impact the performance of the SmartServer; therefore, it is recommended that you use these methods sparingly.

*SetDpProperty(defOutput)*

You can use the `SetDpProperty(defOutput)` method in the `Initialize()` routine of an FPM driver to write the default value of a data point. This is the value that the data point should use when it is not receiving updates, or when it is reset or overridden.

**SYNTAX**

```
void SetDpProperty(FPM::Dp::cfgUCPTdefOutput, int nValue)
```

The `nValue` parameter specifies the default value to be assigned the data point.

**EXAMPLE**

```
F1.SetDpProperty(FPM::Dp::cfgUCPTdefOutput, 100);
```

*SetDpProperty(persist)*

You can use the `SetDpProperty(persist)` method in the `Initialize()` routine of an FPM driver to set whether a data point is persistent (a constant).

**SYNTAX**

```
void SetDpProperty(FPM::Dp::cfgUCPTpersist, bool bValue)
```

The `bValue` parameter specifies whether the data point is persistent. Specify `true` to make the data point a constant. Specify `false` to enable the value of the data point to be updated.

**EXAMPLE**

```
F1.SetDpProperty(FPM::Dp::cfgUCPTdefOutput, true);
```

*SetDpProperty(pollRate)*

You can use the `SetDpProperty(pollRate)` method in the `Initialize()` routine of an FPM driver to set how often the data point is polled.

**SYNTAX**

```
void SetDpProperty(FPM::Dp::cfgUCPTpersist, int nValue)
```

The `nValue` parameter specifies the frequency in which a data point is polled (in milliseconds).

**EXAMPLE**

```
Line1.SetDpProperty(FPM::Dp::cfgUCPTpollRate, 900);
```

*SetDpProperty(unit)*

You can use the `SetDpProperty(unit)` method in the `Initialize()` routine of an FPM driver to specify the unit string used by the data point.

**SYNTAX**

```
void SetDpProperty(FPM::Dp::cfgUCPTunit, const char* const pszValue)
```

The `char` parameter specifies the unit string to be used by the data point.

**EXAMPLE**

```
F1.SetDpProperty(FPM::Dp::cfgUCPTunit, "state");
```

***UFPT Local Variables***

You can use the `DECLARE_FB_INSTANCE_LOCAL()` macro to use additional data point variables that apply to specific functional block instances. For example, you can declare a UFPT local variable that stores how often the `Work()` routine has been called by specific functional block instance or you can declare a UFPT local variable that stores the file name of a functional block instance.

**SYNTAX**

```
DECLARE_FB_INSTANCE_LOCAL(dataType, variableName)
```

**EXAMPLE**

```
// <= section datapoint variable declarations
DECLARE_FB_INSTANCE_LOCAL(int, callCount);
```

---

## External SmartServer Data Point Methods

In an FPM application, you can use the `List()` method with a specific `xSelect` syntax to obtain a list of external data points on the SmartServer. External data points include those data points on the internal SmartServer device [**i.LON App (Internal)**] and the data points of the external devices connected to the SmartServer.

After a data point ID is obtained with the `List()` method, you can use the `Read()` and `Write()` methods in an FPM application to evaluate and update the data point. In addition, you can use the `get()` methods described in the *FPM Application Data Point Property Methods* section in this appendix to read the properties of the external data points on the SmartServer.

**Notes:** Using the `List()`, `Read()`, and `Write()` methods in an FPM application may significantly impact the performance of the SmartServer; therefore, it is recommended that you use these methods sparingly.

### *List()*

#### SYNTAX

```
STATUS List(const string& rsXSelect, FPM::FpmItemColl_t&
rListUniqueIndexes);
```

In the `string` parameter, you specify an `xSelect` statement to be used to filter the external data points on the SmartServer by name. The format used for a data point name is as follows: `<network>/<channel>/<device>/<functional block>/<data point>`. This means that, for example, you can obtain all the data points of the Digital Output 1 functional block on the SmartServer. To do this, you would specify an `xSelect` statement that acquires all unique data points with names starting with "Net/LON/i.LON App/Digital Input 1".

In the `rListUniqueIndexes` parameter, you specify the collection of structures that contain item IDs that are to be used in the `Read` routine.

The `STATUS` value returned by this method can either be `ERROR` or `OK`.

#### EXAMPLE

The following example demonstrates a `List()` method that obtains the data points in the digital output functional block of a lamp that is connected to the SmartServer.

```
FpmItemInfoColl_t items;

if ((List("//Item[starts-with(
UCPTname,\"Net/LON/Lamp/Digital Output/\")]", items)) == OK)
{
    //insert code here
}
```

After the list of the data points in the digital output functional block is acquired, the properties of the listed data points can be obtained using the `get()` data point property methods. The following example demonstrates how to get the properties of the data points returned by the `List()` method.

```
FpmItemInfoColl_t::iterator itEnd = items.end();
for(FpmItemInfoColl_t::iterator it = items.begin();
    it != itEnd; ++it)
{
    CFpmItemInfo &v = (*it);
    printf("UCPTname: %s :: UCPTaliasName: %s :: UCPTindex: [%d]
          :: ItemCfgDepth: %d \n",
```



```

        v.GetDpPropertyAsString(Dp::cfgUCPTname),
        v.GetDpPropertyAsString(Dp::cfgUCPTaliasName),
        v.GetUCPTindex(),
        v.GetDpPropertyAsItemCfgDepth(Dp::cfgItemDepth));
    }

```

**Notes:**

- You can also define an xSelect statement that queries data point properties instead of actual data points. For example, you could define an xSelect statement that queries the **AliasName** configuration property of a data point.

```

char szXSelect[128] =
//Item[@xsi:type="Device_Cfg"][UCPTaliasName="xxo\"];

```

- If you define a more general xSelect statement that could return a set of mixed network objects (e.g., devices, functional blocks, and data points), you need to evaluate the `cfgItemDepth` property of the objects being returned. This property can have one of the following types:

```

ItemCfgDepth_Network      // item is a network
ItemCfgDepth_Channel      // item is a channel
ItemCfgDepth_Device       // item is a device
ItemCfgDepth_Fb           // item is a functional block
ItemCfgDepth_Dp           // item is a data point
ItemCfgDepth_Cp           // item is a configuration property
ItemCfgDepth_UNKNOWN      // item has an unknown object type

```

- You can use the `GetUCPTindex()` in the `List()` method to obtain the network variable index of the data point within its device.

*Read()*

You can perform a `Read()` method in an FPM application to read the values of the data points returned by the `List()` method.

**SYNTAX**

```

STATUS Read(const CFpmItemInfo& rMeta,
            Byte* const pbyValue,
            unsigned int nSize,
            bool bReadProperties = false,
            int nMaxAgeMillis = -1);

```

In the `CFpmItemInfo` parameter, you must specify an object obtained by the `List()` method. The object's `cfgItemDepth` property must be of type `ItemCfgDepth_Fb` or `ItemCfgDepth_Dp`; otherwise, this method returns an ERROR in the STATUS value.

The `pbyValue` parameter specifies the variable used to store the value returned by the method. If a container is provided (`pbyValue != NULL`), this method returns the following properties of the data points in addition to their values (if `bReadProperties` is set to true):

```

FPM::Dp::cfgUCPTname
FPM::Dp::dataUCPTlastUpdate
FPM::Dp::dataUCPTstatus
FPM::Dp::dataUCPTpriority

```

The `nSize` parameter specifies the size of the data point returned by the method.

The `nMaxAgeMillis` parameter specifies how the source of the data point value and how often the FPM application polls the data point. This parameter accepts the following values:

- **-1.** The data point value is read from the SmartServer's internal data server. The poll rate used by the FPM application is set to the poll rate configured for that data point in its **Setup - LON Data Point Driver** Web page. This is the default.
- **0.** The data point value is read directly from the data point. Note that because the data point value is being read synchronously, the FPM application may not be able to perform any other processing until it receives the data point value.
- **>1.** The value you specify is compared to the amount of time that the data point value has been cached in the SmartServer's internal data server.
  - If `nMaxAgeMillis` is less than the period of time the data point value has been cached, the internal data server polls the data point and returns the updated value to the FPM application.
  - If `nMaxAgeMillis` is greater than the period of time the data point value has been cached, the internal data server returns the cached value to the FPM application.

The STATUS value returned by this method can either be ERROR or OK

If you want to read only the values of the data points returned by the `List()` method, you can use a `Read()` method that does not take the `bReadProperties` parameter. In this case, the `Read()` method has the following signature:

```
STATUS Read(const CFpmItemInfo& rMeta,
            Byte* const pbyValue,
            unsigned int nSize,
            int nMaxAgeMillis = -1);
```

#### EXAMPLE

The following example demonstrates a `Read()` method that evaluates the value of a data point returned by a `List()` method.

```
FpmItemInfoColl_t items;
SNVT_time_stamp time;

if(List("//Item[@xsi:type=\"Dp_Cfg\"] [contains(UCPTname,
      \"nvoRtTime Date\"])", items)) == OK
{
    printf("\nitems.size()=%d", items.size());
    if(!Read(items[0], (Byte*)&time, sizeof(SNVT_time_stamp)))
    {
        printf("\nnvoRtTimeDate: %d-%d-%d %d:%d:%d", time.year,
              time.month, time.day, time.hour, time.minute,
              time.second);
    } else printf("\nRead failed");
} else printf("\nList failed");
```

#### *Write()*

You can perform a `Write()` method in an FPM application to write values to the data points returned by the `List()` method.

#### SYNTAX

```
STATUS Write(const FPM::FpmItem& rFpmItem, Byte* pbyValue);
```

In the `rFpmItem` parameter, you must specify an object obtained by the `List()` method. The object's `cfgItemDepth` property must be of type `ItemCfgDepth_Fb` or `ItemCfgDepth_Dp`; otherwise, this method returns an ERROR in the STATUS value.

The `pbyValue` parameter specifies the value to be written to the data point.

The `STATUS` value returned by this method can either be `ERROR` or `OK`.

#### **EXAMPLE**

The following example demonstrates a `Write()` method that changes the value of a data point returned by a `List()` method.

```
FPM::FpmItemColl_t items;
SNVT_time_stamp time;
...
Write(items[0], (Byte*)&time);
```

---

## ***Timer Methods***

You can use timers to perform tasks that must be performed periodically, such as reading data from the RS-232 or RS-485 interfaces or performing data point updates. A separate `CFPM_Timer` application class handles the starting, stopping, and querying of timers.

To use a timer in your FPM, you must first declare it as a member of the `CFPM_Timer` application class in the “Mandatory Application Members” section in the `.h` file using the following syntax:

```
CFPM_Timer m_oTimer1; //declare a timer
```

You then need to initialize the timer in the “Constructor/Deconstructor” section of the `.cpp` file using the following syntax:

```
, m_oTimer1(this) //initialize timer
```

### ***Start()***

You can use the `Start()` method of the `CFPM_Timer` class to start a timer. The `Start()` method causes the system to call back the `OnTimer()` routine, which handles the timer event. The `Start()` method is the standard approach for starting timers. You can use this method in the `Initialize()`, `Work()`, and `OnTimer()` routines of an FPM application, and you can use it in the `Initialize()` routine of an FPM driver.

#### **SYNTAX**

```
void Start(FPM_TimerFlags_t eMode, uint_t nTimeoutMillis);
```

The `eMode` parameter specifies the type of the timer. You can enter `FPM_TF_REPEAT` for a repeating timer, or you can enter `FPM_TF_ONETIME` for a timer that is used just once.

The `nTimeoutMillis` parameter specifies the timer interval in milliseconds. You should set this parameter to a minimum of 100ms.

#### **EXAMPLE**

The following example demonstrates a `Start()` method that starts a timer and repeats it every 2 seconds.

```
CFPM_Timer m_oTimer1; // declared in header file
, m_oTimer1(this) // initialized in source file
...
m_oTimer1.Start(FPM_TF_REPEAT, 2000);
```

### ***START\_TIMER()***

You can use the `START_TIMER()` macro as an alternative approach for starting timers. It causes the system to call back a user-defined timer handler method. You can use the `START_TIMER()` macro

in the `Initialize()`, `Work()`, and `OnTimer()` routines of an FPM application, and you can use it in the `Initialize()` routine of an FPM driver.

### SYNTAX

```
START_TIMER(timeVar, mode, timeoutMillis, funcName)
```

The `timeVar` parameter specifies the name of the timer to be started.

The `mode` parameter specifies the type of the timer. You can enter `FPM_TF_REPEAT` for a repeating timer, or you can enter `FPM_TF_ONETIME` for a timer that is used just once.

The `nTimeoutMillis` parameter specifies the timer interval in milliseconds. You should set this parameter to a minimum of 100ms.

The `funcName` parameter specifies the name of the user-defined timer handler method that is called when this expires.

### TIMER HANDLER SYNTAX

Timers started with the `START_TIMER()` macro must be handled with a user-defined timer handler method that has the following signature:

```
void <funcName>()
```

You must declare your user-defined timer handler method in the “Implements the user functionality” section of the `.h` file.

### EXAMPLE

The following example demonstrates a `START_TIMER()` macro that starts a timer that repeats every 3 seconds and is handled by the `OnMyTimer3()` user-defined timer handler method.

```
CFPM_Timer m_oTimer3; // declared in header file
, m_oTimer3(this)     // initialized in source file
...
START_TIMER(m_oTimer3, FPM_TF_REPEAT, 3000, OnMyTimer3);
...
void OnMyTimer3(); // declared in header file
```

### *Expired()*

You can use the `Expired()` method of the `CFPM_Timer` class to check whether a timer has expired. You can use this method in the `Work()` and `OnTimer()` routines of an FPM application. If this method returns true, you can handle the timer accordingly. For example, you can re-start an expired one-time timer using the `Start()` method.

### SYNTAX

```
bool Expired()
```

If the timer has expired, the method returns `TRUE`. If the timer has not yet expired or if the timer has been stopped, the method returns `FALSE`.

### EXAMPLE

The following example demonstrates an `Expired()` method in an `OnTimer()` routine that checks whether a one-time timer has expired and re-starts it if it has expired.

```
CFPM_Timer m_oTimer2; //declared in header file
void CUFPT_FPM_Application::Initialize()
{
```

```

        m_oTimer2.Start(FPM_TF_ONETIME, 0);
    }
void CUFPT_FPM_Application::OnTimer()
{
    if (m_oTimer2.Expired())
    {
        m_oTimer2.Start(FPM_TF_ONETIME, 2000);
    }
}

```

### *Stop ()*

You can use the `Stop()` method of the `CFPM_Timer` class to stop a timer that is running. You can use this method in the `Work()`, `OnTimer()`, `Shutdown()` routines of an FPM application, and you can use it in the `Shutdown()` routine of an FPM driver.

#### **SYNTAX**

```
bool Stop()
```

If the timer has expired, the method returns `TRUE`. If the timer has not yet expired or if the timer has been stopped, the method returns `FALSE`.

#### **EXAMPLE**

The following example demonstrates how you can use the `Stop()` method to stop a timer that is running.

```
m_oTimer1.Stop();
```

### *StopAllTimers()*

You can use the `StopAllTimers()` method of the `CFPM_Timer` class to stop all currently active timers. You can use this method in the `Work()`, `OnTimer()`, `Shutdown()` routines of an FPM application, and you can use it in the `Shutdown()` routine of an FPM driver.

#### **SYNTAX**

```
bool StopAllTimers()
```

If any timer has expired, the method returns `TRUE`. If no timer has expired or has been stopped, the method returns `FALSE`.

#### **EXAMPLE**

The following example demonstrates a `StopAllTimers()` method that stops all existing timers.

```
StopAllTimers();
```

### *IsRunning()*

You can use the `IsRunning()` method to check whether a timer is running or has been stopped. This method may be useful during runtime.

#### **SYNTAX**

```
bool IsRunning()
```

If the referenced timer is running, the method returns `TRUE`. If the timer has expired or has been stopped, the method returns `FALSE`.

#### **EXAMPLE**

The following example demonstrates an `IsRunning()` method that checks whether a timer is running and executes some code if it is.

```

if (m_oTimer1.IsRunning())
{
    //execute code
}

```

### *GetMode()*

You can use the `GetMode()` method to check the type of the referenced timer (repeating or one-time) when it is running. This method may be useful during runtime.

#### **SYNTAX**

```
FPM_TimerFlags_t GetMode()
```

This method returns the type of timer (repeating or one-time) if the referenced timer is running. If the referenced timer has been stopped, this method returns UNKNOWN.

#### **EXAMPLE**

The following example demonstrates a `GetMode()` method that checks the type of a timer that is running.

```

FPM_TimerFlags_t timerMode;
timerMode = m_oTimer1. GetMode();

```

### *GetTimeoutMillis()*

You can use the `GetTimeoutMillis()` method to check the timeout interval of the referenced timer when it is running. This method may be useful during runtime.

#### **SYNTAX**

```
uint_t GetTimeoutMillis()
```

This method returns the timeout interval (in milliseconds) of the referenced timer if it is running. If the referenced timer has been stopped, this method returns '~0'.

#### **EXAMPLE**

The following example demonstrates a `GetTimeoutMillis()` method that checks the timeout interval of a timer that is running.

```

int timerInterval;
timerInterval = m_oTimer1.GetTimeoutMillis();

```

## ***Reboot Method***

You can use the `fnRebootSmartServer()` method to reboot your SmartServer.

#### **SYNTAX**

```
STATUS fnRebootSmartServer (int a_nBootFlag);
```

This method takes an integer that must be set to 0x00.

#### **EXAMPLE**

The following example demonstrates how to use the `fnRebootSmartServer()` method to reboot your SmartServer.

```

#ifdef __cplusplus
...

```

```

extern "C" STATUS fnRebootSmartServer (int a_nBootFlag);

#endif
...
void CUFPTmath::Work()
{
    printf("\nCUFPTmath::Work(): ");
    if (Changed( in1 ) || Changed( in2 ))
    {
        out3 = in1 + in2;
        printf("out3 = %d = %d + %d", *out3, *in1, *in2 );
    }
    if (Changed(str))
    {
        printf("str = %s", (char*) str->ascii );
    }

    // REBOOT here
    if(100 < *out3)
    {
        fnRebootSmartServer(0x00);
    }
}

```

---

## ***RS-232 Interface Methods***

You can use RS-232 interface methods to connect an FPM driver to the devices attached to the RS-232 serial port on the SmartServer, initialize the RS-232 connection, read and write values to the data points on the devices, and close the RS-232 connection . For more information on connecting a device to the RS-232 serial port, see the *i.LON SmartServer 2.0 Hardware Guide*.

**Note:** The RS-232 interface transmit and receive buffers are both 512 bytes.

### *rs232\_open()*

You can use the `rs232_open()` method in the `Initialize()` routine to open the RS-232 interface on the SmartServer.

#### **SYNTAX**

```
int rs232_open(unsigned int BaudRate);
```

The `BaudRate` parameter specifies the baud rate at which RS-232 interface communicates with the serial port. The SmartServer's RS-232 interface does not support handshakes; therefore, you should select a baud rate that is less than or equal to 19,200. See the documentation for your RS-232 interface for more information on baud rates supported for your device. When setting the baud rate, you should consider the number of bytes the interface sends over the network per second, and the calculations performed between poll cycles.

The method returns the file handle (a value greater than or equal to 0) on success, and it returns a negative value on failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-232 port.

Verify that the file handle is specified in a global variable so that you can reference it from the `Work()` and `Shutdown()` routines.

#### **EXAMPLE**

The following example demonstrates a `rs232_open()` method that opens an RS-232 connection with a baud rate of 9600.

```
int fd = rs232_open(9600);
```

### *rs232\_ioctl()*

You can use the `rs232_ioctl()` method in the `Initialize()` routine to initialize and send commands to the RS-232 interface. You should call this method immediately after opening the RS-232 interface with the `rs232_open()` method. Note that you can send commands to the RS-232 interface using other I/O control methods besides the `rs232_ioctl()` method.

#### **SYNTAX**

```
int rs232_ioctl(int fd, int cmd, int data);
```

The `fd` parameter specifies the file handle that was returned when the RS-232 interface was opened with the `rs232_open()` method.

The `cmd` parameter specifies the command to be sent to the RS-232 interface. The `data` parameter specifies the corresponding value to be used with the `cmd` parameter. The values for the `cmd` and `data` parameters are defined in the **FPMLibrary.h** file in the `iLON/Development/eclipse/plugins/com.echelon.eclipse.ilon100.fpm_0.9.0/compiler/echelon/fpm/include` folder in your LONWORKS directory. The values you can specify for the `cmd` and `data` parameters include the following:

cmd parameter	data parameter
<code>IOCTL_BAUDRATE</code>	Specifies the baud rate at which the RS-232 interface will communicate with the serial port.
<code>IOCTL_RCVBUFSZ</code>	Specifies the receive buffer size.
<code>IOCTL_SIO_HW_OPTS_GET</code>	Gets the current configuration of the hardware options used by the RS-232 interface.

Before using this function, you must call the following function to avoid overriding the existing configuration parameters:

```
int options = rs232_ioctl(fd,  
IOCTL_SIO_HW_OPTS_GET, 0);
```

To set the character size, use the following function:

```
rs232_ioctl(fd, IOCTL_SIO_HW_OPTS_SET,  
((options & ~CSIZE) | CS6));
```

To set odd parity, use the following function:

```
rs232_ioctl(fd, IOCTL_SIO_HW_OPTS_SET,  
(options | PARENB | PARODD));
```

<code>IOCTL_SIO_HW_OPTS_SET</code>	Specify the hardware options for the RS-232 interface.
------------------------------------	--

If the RS-232 interface supports hardware handshakes, enter the following:

```
((options & ~CSIZE) | CS8 | HUPCL | CREAD)
```

If the RS-232 interface does not support hardware handshakes, enter the following:

```
((options & ~CSIZE) | CS8 | HUPCL | CREAD |  
CLOCAL)
```

See the **FPMLibrary.h** file for more information on the values you can pass in as the `data` parameter when the `cmd`



parameter is set to `IOCTL_SIO_HW_OPTS_SET`.

This method returns the 0 upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-232 port.

#### EXAMPLE

The following example demonstrates a `rs232_ioctl()` method that returns the current configuration used by the RS-232 interface.

```
int options = rs232_ioctl(fd, IOCTL_SIO_HW_OPTS_GET, 0);
```

#### *rs232\_read()*

You can use the `rs232_read()` method in the `OnTimer()` routine to read data from the RS-232 interface.

**Note:** Use the `ReadBytes()` function shown below to read data from the RS-232 interface. Do not use the `rs232_read()` function directly to read data from the RS-232 receive buffer because it will cause your FPM Driver to remain in an infinite loop until receive (Rx) data is received from the RS-232 interface.

#### SYNTAX

```
int rs232_read(int fd, unsigned char * buf, int length);
```

The `fd` parameter specifies the file handle returned when the RS-232 interface was opened with the `rs232_open()` method.

The `buf` parameter specifies a pointer to the memory area to where the data read from the RS-232 interface is to be stored. The memory area must have enough space to store the data or else the SmartServer may fail as a result of a call to this method.

The `length` parameter specifies the maximum number of bytes that are to be read.

This method returns the number of bytes read upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-232 port.

#### EXAMPLE

```
rs232_read(fd, someBuffer, 1);
```

**Note:** If there is no data to be read from the RS-232 interface, the call to the `rs232_read()` method will block and wait until new data arrives on the interface. To avoid blocking, you must call both the `select()` and `ioctl( fd, FIONREAD, ... )` methods before calling the `rs232_read()` method in order to set a timeout and therefore avoid blocking.

The following code demonstrates how to do this. The `ReadBytes()` and `BytesReadyForRead()` methods in this example are based on the ones included in the sample RS-232 driver (**Rs232Driver.cpp**) provided in the **LonWorks\iLON\Development\eclipse\workspace.fpm** folder:

```
//define buffer size in header file
#define MAX_RXBUFLLEN 512
Byte rxBuf [MAX_RXBUFLLEN];
...
void CRs232Driver::OnTimer()
{
    if (m_oDisplay_InputTimer.Expired())
    {
        memset(rxBuf, 0, MAX_RXBUFLLEN);
    }
}
```

```

    int nBytesRead = ReadBytes(RS232_fd, rxBuf, MAX_RX_BYTE_SIZE);

    //check whether something has been read
    if (nBytesRead >= 1)
    {
        printf ("Read %c from RS232\n", Line1);
        //if something has been read, write it to display device
        rs232_write(RS232_fd, (Byte *)rxBuf, nBytesRead);
    }
}

int CRs232Driver::ReadBytes(HANDLE handle, Byte* buffer,
                           int bytesToRead) const
{
    fd_set readFds;

    FD_ZERO(&readFds);
    FD_SET(handle, &readFds);

    Timeval readTimeout;
    readTimeout.tv_sec = 0;
    readTimeout.tv_usec = 0; //return immediately if no data is
                             //available

    int nResult = select(handle+1, &readFds, NULL, NULL,
                        &readTimeout);

    if(!nResult || nResult == ERROR)
        return nResult;

    if(!BytesReadyForRead(handle))
        return ERROR;

    return rs232_read(handle, buffer, bytesToRead );
}

int CRs232Driver::BytesReadyForRead(HANDLE handle) const
{
    int bytesAvailable;
    ioctl(handle, FIONREAD, (int) &bytesAvailable );

    return bytesAvailable;
}

```

### *rs232\_write()*

You can use the `rs232_write()` method in the `OnTimer()` routine to write data to the RS-232 interface.

#### **SYNTAX**

```
int rs232_write(int fd, unsigned char * buf, int length);
```

The `fd` parameter specifies the file handle returned when the RS-232 interface was opened with the `rs232_open()` method.

The `buf` parameter specifies a pointer to the memory area containing the data to be written to the RS-232 interface.

The `length` parameter specifies the maximum number of bytes that are to be read.

This method returns the number of bytes read upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-232 port.

#### **EXAMPLE**

The following example demonstrates a `rs232_write()` method that writes to the RS-232 interface.

```
rs232_write(RS232_fd, someBuffer, strlen(someBuffer));
```

#### *rs232\_close()*

You can use the `rs232_close()` method in the `Shutdown()` routine to close the RS-232 interface.

#### **SYNTAX**

```
int rs232_close(int fd);
```

The `fd` parameter is the file handle returned when the RS-232 interface was opened with the `rs232_open()` method.

This method returns the 0 upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-232 port.

#### **EXAMPLE**

The following example demonstrates a `rs232_close()` method that ends an RS-232 connection.

```
rs232_close(fd);
```

---

## ***RS-485 Interface Methods***

You can use RS-485 interface methods to connect an FPM driver to the devices attached to the RS-485 serial port on the SmartServer, initialize the RS-485 connection, read and write values to the data points on the devices, and close the RS-485 connection. For more information on connecting a device to the RS-485 serial port, see the *i.LON SmartServer 2.0 Hardware Guide*.

#### *rs485\_open()*

You can use the `rs485_open()` method in the `Initialize()` routine to open the RS-485 interface.

#### **SYNTAX**

```
int rs485_open(int BaudRate);
```

The `BaudRate` parameter specifies the baud rate at which RS-485 interface communicates with the serial port. See the documentation for your RS-485 interface for more information on baud rates supported for your device. The SmartServer hardware supports connections to any baud rate up to 115,200 with a definable buffer size. The default value is 1,024 bytes. When setting the baud rate, you should consider the number of bytes the interface sends over the network per second, the calculations performed between poll cycles, and whether a hardware handshake between the interface and the hardware device is required.

The method returns the file handle (a value greater than or equal to 0) on success, and it returns a negative value on failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-485 port.

Verify that the file handle is specified in a global variable so that you can reference it from the `Work()` and `Shutdown()` routines.

#### EXAMPLE

The following example demonstrates a `rs485_open()` method that opens an RS-485 connection with a baud rate of 9600.

```
int fd = rs485_open(9600);
```

#### *rs485\_setparams()*

You can use the `rs485_setparams()` method in the `Initialize()` routine to set the operating parameters of the RS-485 interface. You should call this method to initialize the RS-485 interface immediately after opening it with the `rs485_open()` method.

#### SYNTAX

```
int rs485_setparams(int fd, unsigned int BaudRate, EDataLength  
DataLength, EParity Parity, EStopBits StopBits);
```

The `fd` parameter is the file handle returned when the RS-485 interface was opened with the `rs485_open()` method.

The `BaudRate` parameter specifies the baud rate at which RS-485 interface communicates with the serial port.

The `DataLength` parameter specifies the data bit size to be used for messages sent by the RS-485 interface. You can specify a bit size of 5, 6, 7, or 8 bits.

The `Parity` parameter specifies the parity bit to be used for messages sent by the RS-485 interface. A parity bit is an extra bit used to check for errors in groups of data bits transferred between devices. You can specify a parity bit that is odd or even.

The `StopBits` parameter specifies the number of stop bits to be used for messages sent by the RS-485 interface.

For more information on these options, see the **FPMLibrary.h** file in the `iLON/Development/eclipse/plugins/com.echelon.eclipse.ilon100.fpm_0.9.0/compiler/echelon/fpm/include` folder in your LONWORKS directory

This method returns the 0 upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-485 port.

#### EXAMPLE

The following example demonstrates a `rs485_setparams()` method:

```
rs485_setparams(fd, 9600, eDL8Bit, ePNone, eSB1);
```

#### *rs485\_ioctl()*

You can use the `rs485_ioctl()` method in the `Initialize()` routine to send commands to the RS-485 interface.

#### SYNTAX

```
int rs485_ioctl(int fd, int cmd, int data);
```

The `fd` parameter specifies the file handle that was returned when the RS-485 interface was opened with the `rs485_open()` method.

The `cmd` parameter specifies the command to be sent to the RS-485 interface. The `data` parameter specifies the corresponding value to be used with the `cmd` parameter. The values for the `cmd` and `data` parameters are defined in the **FPMLibrary.h** file in the

iLON/Development/eclipse/plugins/com.echelon.eclipse.ilon100.fpm\_0.9.0/compiler/echelon/fpm/include folder in your LONWORKS directory. The values you can specify for the cmd and data parameters are as follows:

cmd parameter	data parameter
IOCTL_BAUDRATE	Specify the baud rate at which the RS-485 interface will communicate with the serial port on the <i>i</i> .LON.
IOCTL_RCVBUFSIZE	Specify the receive buffer size. For more information, see the <code>rs485_setbuffersize()</code> section.
IOCTL_RCVTIMEOUT	Specify the timeout period (in seconds) after which the RS-485 interface stops trying to receive failed messages from the network.
IOCTL_SNDTIMEOUT	Specify the timeout period (in seconds) after which the RS-485 interface stops trying to send a failed messages over the network.

This method returns the 0 upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-485 port.

#### **EXAMPLE**

The following example demonstrates a `rs485_ioctl()` method that sets the buffer size to be used for the RS-485 connection to 256 bytes.

```
rs485_IOCTL(fd, IOCTL_RCVTIMEOUT, 256);
```

#### *rs485\_setbuffersize()*

You can use the `rs485_setbuffersize()` method in the `Initialize()` routine to modify the receive buffer size.

#### **SYNTAX**

```
int rs485_setbuffersize(int fd, unsigned int size);
```

The `fd` parameter specifies the file handle returned when the RS-485 interface was opened with the `rs485_open()` method.

The `size` parameter specifies the new size of the receive buffer to be set. The initial size of the receive buffer is set to 1024.

#### **EXAMPLE**

The following example demonstrates the `rs485_setbuffersize()` method.

```
rs485_setbuffersize(fd, 2048);
```

#### *rs485\_read()*

You can use the `rs485_read()` method in the `OnTimer()` routine to read data from the RS-485 interface.

#### **SYNTAX**

```
int rs485_read(int fd, unsigned char * buf, int length);
```

The `fd` parameter specifies the file handle returned when the RS-485 interface was opened with the `rs485_open()` method.

The `buf` parameter specifies a pointer to the memory area to where the data read from the RS-485 interface is to be stored. The memory area must have enough space to store the data or else the SmartServer may fail as a result of a call to this method.

The `length` parameter specifies the maximum number of bytes that are to be read.

This method returns the number of bytes read upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-485 port.

#### **EXAMPLE**

The following example demonstrates a `rs485_read()` method that reads data from the RS-485 interface.

```
rs485_read(fd, someBuffer, 1);
```

#### *rs485\_write()*

You can use the `rs485_write()` method in the `OnTimer()` routine to write data to the RS-485 interface.

#### **SYNTAX**

```
int rs485_write(int fd, unsigned char * buf, int length);
```

The `fd` parameter specifies the file handle returned when the RS-485 interface was opened with the `rs485_open()` method.

The `buf` parameter specifies a pointer to the memory area containing the data to be written to the RS-485 interface.

The `length` parameter specifies the maximum number of bytes that are to be read.

This method returns the number of bytes read upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-485 port.

#### **EXAMPLE**

The following example demonstrates a `rs485_write()` method that writes to the RS-485 interface.

```
rs485_write(fd, someBuffer, strlen(someBuffer));
```

#### *rs485\_close()*

You can use the `rs485_close()` method in the `Shutdown()` routine to close the RS-485 interface.

#### **SYNTAX**

```
int rs485_close(int fd);
```

The `fd` parameter is the file handle returned when the RS-485 interface was opened with the `rs485_open()` method.

This method returns the 0 upon success, and it returns -1 upon failure. A failure could occur if another FPM is using the interface, or if the interface is not properly connected to the RS-485 port.

#### **EXAMPLE**

The following example demonstrates a `rs485_close()` method that ends an RS-485 connection.

```
rs485_close(fd);
```

---

## File Access Methods

You can read and write to data files on the SmartServer using the following ANSI 'C' file methods: `fopen()`, `fread()`, `fwrite()`, `fseek()` and `fclose()`.

### *fopen()*

You can use the `fopen()` method to open a file and assign it a stream that can be identified by other methods.

#### SYNTAX

```
FILE fopen (const char * filename, const char * mode);
```

The `filename` parameter specifies the name of the file to be opened. This parameter must follow the file name specifications of the Eclipse SDK environment, and it may include a file path.

The `mode` parameter specifies the file access mode, which determines the operations that can be performed on the file stream returned by this method. The file stream can either be a text or a binary file. The mode can be one of the following values:

- `r` Opens an existing text file for reading.
- `w` Creates an empty text file for writing. If a file with the same name already exists, its content is erased and the specified file is treated as a new empty file.
- `a` Appends data to the end of an existing text file. If the specified file does not exist, a new file is created.
- `r+` Opens an existing text file for both reading and writing.
- `w+` Creates an empty text file for both reading and writing. If a file with the same name already exists, its content is erased and the specified file is treated as a new empty file.
- `a+` Opens an existing text file for reading and appending data. All writing operations are performed at the end of the file, protecting the previous content to be overwritten.

You can reposition the internal pointer to anywhere in the file for reading using the `fseek()` method, but writing operations will move it back to the end of file.

If the specified file does not exist, a new file is created.

- `rb` Opens an existing binary file for reading.
- `wb` Creates an empty binary file for writing. If a file with the same name already exists, its content is erased and the specified file is treated as a new empty file.
- `ab` Appends data to the end of an existing binary file. If the specified file does not exist, a new file is created.
- `r+b` Opens a existing binary file for both reading and writing.
- `w+b` Creates an empty binary file for both reading and writing. If a file with the same name already exists, its content is erased and the specified file is treated as a new empty file.
- `a+b` Opens an existing binary file for reading and appending data. All writing operations are performed at the end of the file, protecting the previous content to be overwritten.

You can reposition the internal pointer to anywhere in the file for reading using the `fseek()` method, but writing operations will move it back to the end of file.

If the specified file does not exist, a new file is created.

If the file has been opened successfully, this method returns a pointer to a `FILE` object that is used to identify the stream in all further operations. Otherwise, a null pointer is returned.

## EXAMPLE

The following example demonstrates an `fopen()` method that creates a new text file for writing and assigns it to a file stream.

```
FILE * pFile;  
pFile = fopen ("myfile.txt", "w");
```

## *fread()*

You can use the `fread()` method to read a block of data from a file stream. This method reads an array of elements from the file stream and stores the data in a block of memory on the SmartServer.

**Note:** The SmartServer does not have memory protection. As a result, an FPM can write to any memory area on the SmartServer, which could cause the SmartServer to fail or corrupt the data of another embedded applications on the SmartServer. Therefore, you should use the greatest possible care when implementing your freely programmable modules.

## SYNTAX

```
size_t fread (void * ptr, size_t size, size_t count, FILE *  
stream);
```

The `ptr` parameter specifies the block of memory to be used to store the data read from the file stream with a minimum size of (*size\*count*) bytes.

The `size` parameter specifies the size (in bytes) of each element to be read.

The `count` parameter specifies the number of elements to be read, each one with a size of *size* bytes.

The `stream` parameter specifies a pointer to a `FILE` object that specifies an input stream.

This method returns a `size_t` object, which is an integral data type that specifies the total number of elements successfully read. If this number differs from *count*, either an error occurred or the internal End-of-File internal indicator was reached.

## EXAMPLE

The following example demonstrates an `fread()` method.

```
FILE *pFile;  
int iTemp;  
long FilePtr = 0; // starting index of file  
char buf[200]; // input buffer  
char len = 200;  
char Filename[] = "/web/user/mUartTxFile.txt";  
  
if((pFile = fopen(Filename, "r")) == NULL)  
{  
    printf("Can't open i.LON file = %s \n", Filename);  
}  
else  
{  
    iTemp = fseek(pFile, FilePtr, SEEK_SET);  
    if(iTemp)  
    {  
        printf("FSEEK failed\n");  
    }  
    else  
    {  
        iTemp = fread(buf, 1, len ,pFile);  
    }  
}
```



```

        fclose(pFile);
    }
}
fseek()

```

You can use the `fseek()` method to set the position indicator associated with the a file stream to a new position.

#### SYNTAX

```
int fseek (FILE * stream, long int offset, int origin);
```

The `stream` parameter specifies a pointer to a `FILE` object that identifies the stream.

The `offset` parameter specifies the number of bytes to be offset from *origin*.

The `origin` parameter specifies the position from where *offset* is to be added. You can specify the `origin` using one of the following three constants:

<code>SEEK_SET</code>	Beginning of the file
<code>SEEK_CUR</code>	Current position of the file pointer
<code>SEEK_END</code>	End of the file

If this position indicator has been moved successfully, this method returns a zero value. Otherwise, it returns a non-zero value.

#### EXAMPLE

The following example demonstrates an `fseek()` method that moves the position indicator two bytes from the beginning of an existing file.

```
FILE * pFile;
fseek (pFile , 2 , SEEK_SET);
```

*fwrite()*

You can use the `fwrite()` method to write an array of elements from a block of memory on the SmartServer to the current position in a file stream.

#### SYNTAX

```
size_t fwrite (const void * ptr, size_t size, size_t count, FILE *
stream);
```

The `ptr` parameter specifies the array of elements to be written to the file stream.

The `size` parameter specifies the size (in bytes) of each element to be written.

The `count` parameter specifies the number of elements to be written, each one with a size of *size* bytes.

The `stream` parameter specifies a pointer to a `FILE` object that specifies an output stream.

This method returns a `size_t` object, which is an integral data type that specifies the total number of elements successfully written. If this number differs from *count*, an error has occurred.

#### EXAMPLE

The following example demonstrates an `fwrite()` method that writes some data to an existing text file.

```
FILE *pFile;
char Filename[] = "/web/user/mUartTxFile.txt";
char buf[] = "Test String"; // output buffer
long len = 11;
```

```

if((pFile = fopen(Filename,"a")) == NULL)
{
    printf("Can't open or create i.LON 100 file = %s \n",
        Filename);
}
else
{
    fwrite(buf,1,len,pFile);
    fclose(pFile);
    printf("Wrote %d bytes to file = %s \n", len, Filename);
}
}
fclose()

```

You can use the `fclose()` method to close a file.

### **SYNTAX**

```
int fclose (FILE * stream);
```

The `stream` parameter specifies a pointer to a `FILE` object that specifies the file stream to be closed.

If the file has been closed successfully, this method returns a zero value. Otherwise, it returns EOF (End-of-File).

### **EXAMPLE**

The following example demonstrates an `fclose()` method that closes a file stream.

```
FILE * pFile;
fclose (pFile)
```

## Appendix B

# FPM Development and Deployment Checklist

This checklist outlines the steps required to develop and deploy your FPMs on the SmartServer.

<b>1. Create User-Defined Functional Profile Template (UFPT)</b>		<b>NodeBuilder Resource Editor</b>
<input type="checkbox"/>	Create a new FPM resource file set for company.	Request temporary manufacturer ID from LonMark <a href="http://www.lonmark.org/mid">www.lonmark.org/mid</a> if you do not have one or, if your company has many FPM developers.  Must create scope 5 resource file set if integrating FPM application with an LNS Application such as the LonMaker tool.
	Create a new functional profile template or create one that inherits from an existing Standard Functional Profile Template (SFPT).	
<input type="checkbox"/>	Add NVs and CPs to the UFPT which your FPM will read and write.	
<input type="checkbox"/>	Generate company's FPM resource file set	
<input type="checkbox"/>	Upload company's updated FPM resource file set to root/lonWorks/types/User/<YourCompany> folder on SmartServer flash disk.	
<b>2. Create Device Interface (XIF) File</b>		<b>Text Editor (e.g. Notepad) LonWorks Interface Developer</b>
<input type="checkbox"/>	Use text editor such as Notepad to create a model file (.nc extension).	
<input type="checkbox"/>	Use <i>i.LON</i> SmartServer 2.0 LonWorks Interface Developer tool convert model file to device interface (XIF) file.	
<input type="checkbox"/>	Upload XIF file to root/lonWorks/Import/<YourCompany> folder on SmartServer flash disk.	
<b>3. Write FPM Application or Driver</b>		<b><i>i.LON</i> SmartServer 2.0 Programming Tool</b>
<input type="checkbox"/>	Create new FPM project from the UFPT.	
<input type="checkbox"/>	Write and build FPM application or driver.	Must have full version of <i>i.LON</i> SmartServer 2.0 Programming Tools to build FPM.  The data points declared in the FPM must be in the UFPT.
<b>4. Deploy FPM Application or Driver on Development SmartServer</b>		<b><i>i.LON</i> SmartServer 2.0 Programming Tool <i>i.LON</i> SmartServer 2.0 LonMaker Tool <i>i.LON</i> Vision 2.0</b>

□	Use <i>i</i> .LON SmartServer 2.0 Programming Tool to upload FPM executable module (. <b>app</b> or . <b>drv</b> extension) to root/modules/user/< <i>YourCompany</i> > folder on SmartServer flash disk.	Must have FPM license on SmartServer to upload FPMs.  Select <b>Default Web Page</b> check box in Deployment Settings window of <b>Install FPM Module</b> dialog to create custom FPM configuration Web page for FPM applications.
□	If you are deploying an FPM driver, reboot the SmartServer.	
□	Use SmartServer to verify that a network management service has been selected (LNS or Standalone).	If using LNS mode ( <b>LNS Auto</b> or <b>LNS Manual</b> ), FPM devices must have static interfaces.  Must use an LNS mode in order to bind data points in your FPM application to other data points with LONWORKS connections.  Must use Standalone mode if FPM devices use dynamic interfaces.
□	Use SmartServer to add new internal FPM device on LONWORKS channel.	If FPM device has static interface, select XIF file from root/lonWorks/Import/< <i>YourCompany</i> > folder.  If FPM device has dynamic interface, select v40 XIF file from root/lonworks/import/Echelon/iLON100 folder.
□	If FPM device has dynamic interface, add functional block to device based on UFPT used by FPM.	
□	If binding FPM data points with LONWORKS connections, commission FPM device using SmartServer or LonMaker tool	Can commission device from SmartServer tree or LNS tree in SmartServer Web interface.  Once FPM device is commissioned in the LonMaker tool, cannot use the SmartServer to decommission the device or set the device application offline.  Can only use the LonMaker tool to decommission and re-commission the device and set the device application online or offline.
□	Test FPM application with <b>View – Data Points</b> Web page in SmartServer Web interface.	
□	Connect FPM data points with LONWORKS connections or Web connections.	Can create LonWorks connections with LNS tree in SmartServer Web interface or the LonMaker tool.  Can create Web connections with SmartServer tree in SmartServer Web interface

□	Use Adobe Contribute CS3 with <i>i.LON</i> Vision 2.0 toolkit to create custom FPM configuration Web Pages.	To view custom FPM configuration Web page, click <b>General</b> above navigation pane on left side of the SmartServer Web interface and then click a functional block under FPM device.
<b>5. Deploy FPM Applications or Drivers on Multiple SmartServers in Field</b>		<b>FTP Client such as Internet Explorer 7</b>
	Use FTP client to copy the following files to the folder listed in the column to the right on each SmartServer:	Each SmartServer must have FPM license installed on it.
□	FPM License (if not pre-installed on SmartServer).	root/config/license
□	Resource files (.ENU, .fmt, .fpt, .ls, and .typ files).	root/lonworks/types/user/<YourCompany>
□	Device interface (XIF) files (.xif extension)*. *If the FPM application uses static functional blocks.	root/lonworks/import/<YourCompany>
□	FPM executable modules (.app or .drv extension).	root/modules/user/<YourCompany>
□	Custom FPM configuration Web pages (.htm extension)* for FPM application. *If created.	root/web/config/Fb

# Appendix C

## FPM FAQ

This FAQ answers common question related to FPMs.

---

# SmartServer FPM FAQs

## 1. What are Freely Programmable Modules (FPMs)?

FPMs are custom C/C++ applications or drivers (RS-232 or RS-485) that allow you to customize the SmartServer's embedded software. You can use FPMs to perform tasks not provided by the SmartServer's built-in applications. FPMs can have both network variables (NVs) and Configuration Properties (CPs).

- An **FPM Application** reads and writes values to the data points declared in it. An FPM application executes code upon data point updates, reads data point properties, and controls timers and executes code upon their expiration. For example, you can create an FPM application that takes two input NVs, adds their values when either one changes, and then writes the value to an output NV ( $in1 + in2 = out$ ).
- An **FPM Driver** provides values for the data points declared in it by reading and writing to the RS-232 and RS-485 ports on the SmartServer. FPM drivers let you create gateways for non-native devices.

## 2. What do I need to create FPMs?

You need to purchase the *i.LON SmartServer 2.0 Programming Tools DVD* (Echelon Model 72111-409). The *i.LON SmartServer 2.0 Programming Tools* includes a pre-configured Eclipse Development Kit that you can use to write, build, and upload FPMs to your SmartServer.

Only FPM developers need this tool. That is, you don't need this tool to use an FPM developed by someone else.

## 3. How do I deploy FPMs on my SmartServer?

- Your SmartServer must have the following files to deploy an FPM application or driver:
  - FPM Programming Activation Key. Each SmartServer needs its own unique license file, so don't copy this file to all SmartServers. If your SmartServer does not have a Programming Activation Key, you can order one (Echelon Model 72161) at [www.echelon.com/ilon/activate](http://www.echelon.com/ilon/activate)
  - Optional: FPM Application License. An optional license created by the FPM developer.  
<i.LON root directory>/config/license
  - User-defined resource file set (\*.ENU, \*.fmt, \*.fpt, \*.ls, and \*.typ files), which defines the functional profile template used by your FPM application or driver (that is, the FPM NVs and CPs).  
<i.LON root directory>/lonworks/types/user/<YourCompany>
  - FPM executable module (.app or \*.drv extension).  
<i.LON root directory>/modules/user/<YourCompany>
  - Optional: FPM web pages (\*.htm)  
<i.LON root directory>/web/config/Fb
    - Optional: NLS local language files (e.g., German) for the FPM web page.  
<i.LON root directory>/web/config/Fb/nls
  - Optional: External device interface (XIF) file is required if you are deploying an FPM application that has a static interface (uses static functional blocks)  
<i.LON root directory>/lonworks/import/<YourCompany>



- For FPM Applications.
  - Add an internal device using the above XIF (Static FBs) or the SmartServer's v40 XIF (Dynamic FBs).
  - You may need to commission the internal device in order to use certain features (e.g., when using LONWORKS connections with Static FBs)
- For FPM Drivers.
  - Nothing, once uploaded to the SmartServer you are done.

#### 4. Do I need to re-compile my FPM code when I install a SmartServer Service Pack or Upgrade?

For most SmartServer service packs (SP) and upgrades (U), you do not have to re-compile your FPMs.

The exceptions may be that you may want to use a new feature associated with the new service pack/upgrade or there is a change to the SmartServer embedded firmware that affects FPMs. After each service pack or upgrade, check the FPM ReadMe to see if your FPM code needs to be compiled or rebuilt.

Additionally, the SmartServer will report a console port error during the boot process if the FPM build version doesn't match the SmartServer version. The FPM build version is the SmartServer version used when compiling the FPM. For example, if you compiled the FPM with SmartServer version 4.03, then the FPM build version is 4.03. During a reboot (or if you are using the *i.LON* SmartServer 2.0 Programming Tool to download an FPM to the SmartServer), the SmartServer will check the FPM build version to see if it is supported with the current SmartServer version. If there is a problem, the SmartServer will then print an error message to the SmartServer console port.

#### 5. What are internal devices?

An internal (virtual) device is an emulation of a device that resides on the SmartServer. An internal device encapsulates one or more functional blocks, which are instances of the FPM applications you have created with the *i.LON* SmartServer 2.0 Programming Tools. You can create up to 10 internal devices on the SmartServer that can be used for FPM applications.

For Static Functional Blocks (FBs), you must create a XIF file for the internal device with the LonWorks Interface Developer Tool LIBILON. The LIBILON tool limits the XIF Address Table entries to 15. See instruction later in this FAQ on how to increase this value.

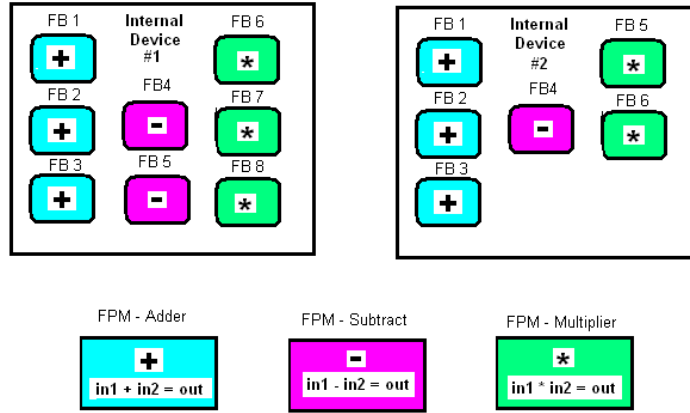
#### 6. Can an internal FPM device have multiple applications (or FBs)?

Yes, a single internal device can have multiple FBs based on multiple FPMs.

For example, if you create three FPMs with each FPM using 2 input NVs and one output NV:

FPM 1 - Adder:  $in1 + in2 = out$ ,  
 FPM 2 - Subtract:  $in1 - in2 = out$ ,  
 FPM 3 - Multiplier:  $in1 * in2 = out$

You can then add an internal device to the SmartServer (see internal device #1 in the figure below) in which there are 3 FBs based on an Adder FPM, 2 FBs based on a Subtraction FPM and 3 FBs based on a Multiplier FPM. You can add additional internal devices with a different number and types of FBs (see internal device #2).



If you use Static FBs for the figure above then you would have to create two XIF files because the number of FBs is not the same for both devices. If you were using multiple identical internal devices then you only need to create one XIF file.

**7. What is the relationship between FPM applications and the functional blocks (FBs) in an internal device?**

The relationship is similar to the built-in SmartServer applications and the functional. The SmartServer’s built-in applications (also called modules) perform a specific task such as alarm notification, scheduling, or data logging. A functional block is an instance of a specific application. For example, to use the SmartServer’s Alarm Notifier application, you first need to add an Alarm Notifier functional block to the SmartServer’s i.LON App device.

Your FPM application performs a specific task, and to instantiate an FPM application, a functional block representing an instance of the FPM application must be added to the internal FPM device.

- If the internal device uses static FBs then you specify the XIF file to be used by the internal FPM device when you add the device. All the functional blocks (FPM application instances) and data points specified by the XIF file will automatically appear under the internal FPM device in the navigation tree on the left side of the SmartServer Web interface
- If the internal device has dynamic FBs then you manually add functional blocks to the internal FPM device for the FPM applications to be instantiated.

**8. Does an internal device need to use FPM static or dynamic Functional Blocks?**

The type of FB that you need to use depends on how you are going to use the FB and the type of connection (LonWorks connections vs. Web connections) use for the FB NVs/CPs.

For static FBs you need to use an XIF file when creating and internal device. Static FB can be used with LonWorks connections or web bindings and need to be added to the LON channel.

For dynamic FBs, you manually add the FB to an internal device. Dynamic FB can only be used with web bindings and should be added to the SmartServer virtual channel.

For Standalone mode you can use either static or dynamic FBs and the internal device can be added to either the SmartServer virtual channel or LON channel.

For LonMaker (or other LNS Application), the location or type of FB depends on your application. To use LonWorks binding with LonMaker or an LNS Application then you will need to use static FBs. If you are only using web bindings then it may make more sense to use dynamic FBs.

For Static FBs, you need to create a model file (\*.nc extension) in which you declare all the data points in the UFPT used by the FPM, and a functional block that implements an instance of the UFPT. You then need to use the *i.LON SmartServer 2.0 LonWorks Interface Developer* tool to convert the model file to a XIF file. When you create the internal device on the SmartServer, all the functional blocks and data points specified by the XIF file will automatically appear under the internal FPM device in the navigation pane on the left side of the SmartServer Web interface.

One advantage of using static FBs is that once you create the XIF file, it is easy to add new internal FPM devices with the same feature set. A disadvantage of using static FBs is that they consume resources on the SmartServer even if a FB or NV/CP is not being used.

If you are running your network with the SmartServer operating as a standalone network manager, the internal device can use static or dynamic FBs. If the internal FPM device uses dynamic FBs then you need to create an internal device that uses the SmartServer internal v40 XIF file. You then add functional blocks to the internal FPM device for the FPM applications you want to instantiate.

One advantage of using dynamic FBs is that you only use SmartServer resources for those functional blocks that you add. A disadvantage of using dynamic FBs is that when you want to add another internal device with the same feature set, you must manually add all the FBs again.

#### **9. Do I need to commission an internal FPM device?**

You do if you plan on using LONWORKS connections to bind the data points in your FPM application to data points on the SmartServer or to data points on external devices. You do not need to commission an internal FPM device if you plan on connecting FPM data points with Web connections (Web bindings).

Note that LonWorks connections are only supported in **LNS Auto** or **LNS Manual** mode. If you are operating the SmartServer in **Standalone** mode, you can only use Web connections to bind your FPM data points.

#### **10. How do I create unique names for my FPMs to avoid collisions with other FPM manufacturers?**

Each FPM must be identified by a unique namespace that consists of a Program ID and a Functional Profile Template name (e.g., “8FFFFFF46140A1E03[5].UFPTMath”). That is, no two FPMs on the same SmartServer can have the same namespace. Using your company’s program ID in the namespace prevents potential naming conflicts.

You can use Scope 3, 5 or 6. Scope 5 is recommended. If you don’t have a Manufacturer ID then go to the LonMark website [www.lonmark.org/mid](http://www.lonmark.org/mid) and get a temporary Manufacturer ID.

#### **11. Can I create network variables (NVs) for my FPMs?**

Yes. You can create both input and output NVs for your FPM. You need to define your NVs in a resource file set using the NodeBuilder Resource File Editor. The NVs are automatically declared in your FPM code when you create a new FPM project based on a UFPT in your resource file set.

#### **12. Can I use configuration properties (CPs) for my FPMs?**

Yes, you can create CPs that are implemented as NVs. You need to define your CPs in a resource file set using the NodeBuilder Resource File Editor. The CPs are automatically declared as CP NVs in your FPM code when you create a new FPM project. If you plan on using static FBs for your internal FPM devices, you also need to define the CPs as CP NVs in the model file.

CPs that are implemented in configuration files are not supported.

#### **13. What is the NodeBuilder Resource File Editor and why do I need it?**

The NodeBuilder Resource Editor is used to develop a user-defined functional profile template

(UFPT), which defines the NVs and CPs to which your FPM will read and write. The NodeBuilder Resource Editor is available on the *i.LON SmartServer 2.0 Programming Tools DVD*. The UFPT serves as the FBs specification for your FPM.

#### **14. What is a resource file set?**

These files define the external FBs of your FPM module. This includes the NVs and CPs to which your FPM will read and write. The resource files set needs to be defined before you can start coding your FPM. If the UFPT to be used by your FPM inherits from a standard functional profile template (SFPT) then you don't need to do anything. In many cases your FPM will have a custom look and will require you to generate a new resource file set using the NodeBuilder Resource Editor.

A resource file set is company specific. If you don't already have a LonMark company ID, you can apply for one at [www.lonmark.org/mid](http://www.lonmark.org/mid).

After you create a resource file set, you need to FTP it to your SmartServer and then reboot the SmartServer. Your new resource file set is located under the **lonWorks\types\user\<manufacturer name>** folder on your computer. Copy the resource file set (the \*.enu, \*.fmt, \*.fpt, \*.is, and \*.typ files) to the **root/lonWorks/types/user/<manufacturer name>** folder on the SmartServer flash disk. After you copy your resource file set to the SmartServer, reboot the SmartServer so that the SmartServer can use the new/modified files.

#### **15. What is a Functional Profile Template (FPT)?**

A functional profile template contains definitions of all the NVs and CPs that your FPM supports. Your FPM must have a UFPT (user-defined FPT) that is defined in a Resource File Set. The UFPT is created using the NodeBuilder Resource Editor.

#### **16. Can I create an FPM based on a Standard Functional Profile Template (SFPT)?**

No. You can create a User Functional Profile Template (UFPT) that inherits from an existing SFPT or you can create a new UFPT from scratch.

#### **17. What is a Model File?**

A model file is used to create an XIF file for internal devices that use Static FBs. This is not needed for Dynamic FBs. A XIF can only be made up of FPMs from the same manufacturer.

A model file uses the Neuron C programming language to describe the functional blocks, network variables, and configuration properties in an FPM application. You do not need to be proficient in Neuron C to create a model file for an FPM because the model file does not include executable code.

#### **18. What is the *i.LON SmartServer 2.0 LonWorks Interface Developer Tool LIBILON.EXE (Static FBs only)*?**

The *i.LON SmartServer 2.0 LonWorks Interface Developer Tool* is a command line interface that converts a model file (\*.nc extension) to a static device interface (XIF) file. If you are integrating your FPM application with the LonMaker tool, LNS tree, or another LNS application, your internal FPM device must have a static interface.

This tool limits the XIF to 15 entries in the SmartServer internal device Address Table. This means that an internal device based on this XIF file can have LonWorks Connection to only 15 different devices. Once you create the XIF file you can modify the file to support more than 15 address Table entries using the instructions below.

#### **19. How can I change the XIF file to support more than 15 Address Table entries per Internal Device (Static FBs only)?**

You can modify the XIF file with a text editor like Notepad or WordPad (use text only). You can modify both the address table entries and alias table entries.

File: fpmMathAdder.xif generated by LonTalk Interface Developer Revision 3.00.35, XIF Version 4.401

Copyright (c) Echelon Corporation 2002-2008

All Rights Reserved. Run on Tue Apr 29 16:48:44 2008

8F:FF:FF:00:00:00:00:02

2 15 0 20 0 4 4 2 2 4 4 0 0 0 5 0 16 1024 1 1 64 20 0 1 3 193 0 0 0 0 0 2 4096 0 0 0 0 0 2 1 0 0

128 5 128 0 0 0 0 15 5 8 0 10 10000000

1 7 1 1 4 4 4 15 200 0

78125 0 0 0 0 0 164 0 0 0 0 0

90 0 240 0 0 0 40 40 0 5 8 5 12 14 15

- **Number of address table entries:**  
34 entry on line 6 of the XIF file; change 15 to 4096 in the example above.
- **Number of alias table entries:**  
19 entry on line 6 of the XIF file; change 0 to 1024 in the example above.

**20. Can I use FPMs to modify the SmartServer's built-in applications (e.g., Alarm Generator, Scheduler)?**

No, you can't modify the SmartServer's built-in functions, but you can access the Built-in applications' data points.

**21. Can the FPM module directly access data points of SmartServer built-in Applications (e.g., Alarm Notifier)?**

Yes, you can use the List(), Read(), and Write() methods to access NVs that are external to your FPM. However, these methods significantly impact the performance of the Smart Server. It is recommended that you access the data points on the SmartServer by connecting them to data points in an FPM application with LONWORKS and Web connections.

**Note:** If the SmartServer is operating in **LNS Auto** or **LNS Manual** mode, you can use LONWORKS and Web connections. If the SmartServer is operating in **Standalone** mode, you can only use Web connections.

**22. Can the FPMs call or access other SmartServer applications such as an Alarm Generator?**

No, an FPM can not call any of the SmartServer's built-in applications. To access the data points in the SmartServer's built-in applications, you can use the FPM List(), Read(), Write() methods; LonWorks connections; and Web connections.

**23. Can an FPM dynamically create data points in an LNS network database?**

No. Only the data points that you declare in the UFPT used by your FPM are added to the LNS network database.

**24. Can I use Changeable Types for my FPM NVs?**

No. You can only use one NV type (SNVT or UNVT) for each FPM NV which you specify within the resource file set.

**25. Can I add Dynamic NVs to my FPM FB?**

No. Dynamic NVs are not supported (only NVs that are declared at compile time are supported).

**26. Can I access the Configuration Properties of other devices?**

Only CPs that are defined as CP NVs.

**27. Does an FPM have access to the SmartServer TCP/IP protocol stack (Sockets)?**

No, this feature is not supported. FPMs can not directly make an outbound SOAP call. Nor can the FPM be referenced directly by inbound SOAP call.

**28. Can I read and write files on the SmartServer?**

Yes. You use C/C++ file methods to access files on the SmartServer. Additionally, you can use a SOAP application or FTP on a PC to download/upload files from the SmartServer.

**29. Can I protect my FPM applications from piracy?**

Yes. You can add code to your FPM application that checks whether a SmartServer has a FPM Application license (that you create) in order to run your FPM. The *i.LON* SmartServer 2.0 Programming Tools includes an *i.LON* License Generator program that you can use to create licenses for your FPM applications that must be on a SmartServer to run your FPM application. See Chapter 7 for more information on creating FPM application licenses.

**30. Can developers create timed demo License (for example, 30 days)?**

There is no direct support for this feature, but you could code your FPM to support it.

**31. Is there a debugger for the FPM Development kit?**

Echelon does not provide a debugger; however, you can use printf() statement to print out debugging information to the SmartServer console port. FPMs are based on a VxWorks operating system; therefore, you can purchase a source level debugger (VxWorks 6.2 - Wind River Workbench 2.4). Contact Wind River® sales at [www.windriver.com/company/contact/index.html](http://www.windriver.com/company/contact/index.html) for more information on ordering “WindRiver Platform for Industrial Services V3.2 for MIPS32 Processors”.

**32. Can I run my FPM application on the *i.LON* 100 hardware (e3 or prior version)?**

No. FPMs are only supported on the SmartServer hardware.

**33. How many FPM application modules can I load to a SmartServer at a time?**

There is no defined limit. However, there is a practical limitation due to the memory size (64MB flash) which is used for all SmartServer applications and data points.

**34. How many FBs can I use for a given FPM application?**

There is no limit. You can add as many FPM FBs to a single internal device as you want and you can add up to 10 internal devices to a SmartServer. However, there is a practical limitation due to the memory size (64MB flash) which is used for all SmartServer applications and data points.

**35. How many data points (NVs or CPs) can I create for one FPM application module?**

There is no limit for the FPM module.

Echelon recommends that you add no more than 1,000 data points to the SmartServer. This includes

data points on external devices, dynamic network variables added to the SmartServer's internal automated systems device ["i.LON App (Internal)" by default], and data points in FPM applications and drivers.

You can add more than 1,000 data points, but you should periodically open the **Setup - System Info** Web page to verify that SmartServer has enough resources (Spare Flash Blocks, Free Disk Space, RAM, and CPU Utilization) to support the additional data points.

The amount of resources consumed by the SmartServer depends on the types and number of applications running on the SmartServer (a Data Logger can consume a lot of resources). Note that the static data points in the SmartServer's built-in applications/ functional blocks do not count against the 1,000-data point limit

**36. Can I access NV elements in a structured NV from FPM application module? (e.g., "value" element in SNVT\_switch type NV)?**

Yes, structured NVs type fields or elements can be accessed directly or through temporary data point variables. See Chapter 5 for more information on reading and writing to structured data points.

**37. Does the FPM have access to the source address of an input NV or can it determine how many LONWORKS or Web connections are connected to an input NV?**

No.

**38. How fast can the FPM Timer run?**

We recommend no faster than 100 ms.

**39. Does the FPM provide support for LonTalk Explicit Messages?**

No.

**40. Does the FPM have access to data point status information (for example, if the data point is online)?**

Yes, the FPM has read access to data point name, alias name, status, write priority, and last update. See Chapter 5 for more information on reading these data point properties.

**41. Can I determine which FB instance the FPM is currently running?**

The best way to get the information is to look at the name property of one of the NVs that your are using. Since the NVs and CPs are local to a FB instance, you can determine the instance index number from the name property.

The first step is to pick one of your NVs or CPs that you are using. Next get the full path name for the NV/CP. From the return data strip out the FB index number.

In the example below, NV nviTemp is used to determine the FB index. FB index is then saved in a local variable. This local variable is local to the FB instance. That is, no other instance of the FB can access this variable. In this case local variable **iFbNumber** is used. Note local variables are declared using a UFPT local variable.

```
// => section datapoint variable declarations. DO NOT REMOVE THIS SECTION'S COMMENT!  
DECLARE( _0000000000000000_0_::SNVT_temp_p, nviTemp, INPUT_DP )  
// <= section datapoint variable declarations. DO NOT REMOVE THIS SECTION'S COMMENT!  
//-----  
// My Code - Begin -----  
DECLARE_FB_INSTANCE_LOCAL(int, iFbNumber); // LocalVariable  
// My Ends - Begin -----  
//determine the FB index in the initialization routine
```

```

void CUFPTfpmTempController::Initialize()
{
// Determine FB index by looking at one of the NVs name, [index]
    char * nv_name =
        (char *)nviTemp.GetDpPropertyAsString(FPM::Dp::cfgUCPTname);
    // get NV name

    // Get FB index number from name
    char * tmp_name;
    tmp_name = strtok(nv_name, "[");
    tmp_name = strtok(NULL, "]");
    iFbNumber = strtol(tmp_name, NULL, 10); // convert String to int

    printf("UFPTfpmTempController[%i]::Initialize() \n", *iFbNumber);
    //enter comment in work routine to indicate when nviTemp is changed

void CUFPTfpmTempController::work()
{
    if(Changed(nviTemp))
    {
        printf("UFPTfpmTempController[%i]::Work() - nviTemp =
            *iFbNumber, *nviTemp);
    }
}
}

```

#### 42. What is the difference between Local and Global FPM variables?

Local variables are defined and used differently than global variables.

- FPM local variables are local to a FB instance.
  - NVs and CPs are always local variables.
  - A local integer variable named iFbNumber would be defined as:
 

```
DECLARE_FB_INSTANCE_LOCAL(int, iFbNumber); // local variable
```

 Each FB instance will have its own iFbNumber that only it can read or write.
- FPM global variables are accessible by all FB instances.
  - All FB instances for a FPM can read and write to this variable.
  - FPM global variables are defined as normal C/CC+ variables and are used by all FB instances of the FPM. A global integer variable named iCount would be defined as:
 

```
int iCount; // global variable
```
- You should use local variables whenever the data is FB instance related (e.g., FB instance index number, state information, timer information).
- Always design your FPM assuming that it will be used with multiple FBs.
- If you see erratic behavior from a FB, this may mean that you are using a global variable when you should be using a local variable.
- See the Echelon Knowledge Base web site for examples.

#### 43. Can a FPM driver have NVs and CPs?



Yes. This how you pass data from the FPM driver to the FPM applications.

**44. Can a FPM application access the default UART Drivers?**

No, you will need to write your own driver.

**45. Can you use logic in a FPM driver?**

You can, but we don't recommend it. You should use the FPM driver to send and receive data to the UART and do all of your processing in a FPM application.



[www.echelon.com](http://www.echelon.com)