Fieldbus

NI-FBUS™ Communications Manager User Manual

July 2000 Edition
Part Number 321287C-01

Worldwide Technical Support and Product Information

www.ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Conventions

The following conventions are used in this manual:

»	The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
Fieldbus	The generic term *Fieldbus* refers to any bus that connects to field devices. This includes FOUNDATION Fieldbus, CAN, DNET, and Profibus. In this manual, the term *Fieldbus* refers specifically to the FOUNDATION Fieldbus.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Contents

Chapter 1
Introduction

Chapter 2
Developing Your Application

Chapter 3
NI-FBUS Dialog Utility

1

Introduction

This manual describes and explains how to use the NI-FBUS Communications Manager software. This manual assumes that you are already familiar with your Microsoft operating system.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- *Foundation Specification: System Architecture*
- *Foundation Specification: System Management*
- *Foundation Specification: Network Management*
- *Foundation Specification: Fieldbus Message Specification*
- *Foundation Specification: Function Block Application Process, Parts 1 and 2*
- *Foundation Specification: Device Description Services User Guide*
- *Foundation Specification: 31.25 kbit/s Physical Layer Profile*
- *Foundation Specification: Device Description Services User Guide*
- *Device Description Language Specification*

NI-FBUS Communications Manager Overview

The NI-FBUS Communications Manager implements a high-level Application Program Interface (API) that lets you communicate with the National Instruments FOUNDATION Fieldbus communication stack and hardware. The main purpose of the NI-FBUS Communications Manager is to make the details of the Fieldbus communication protocols transparent by offering you an API that supports `TAG.PARAMETER` access. You need a general knowledge of the Fieldbus architecture (outlined in the *FOUNDATION Fieldbus Overview* document) to understand and use the NI-FBUS Communications Manager.

The NI-FBUS Communications Manager handles communication between the communication stack and the user application. It also handles the details

of communicating with the Fieldbus Messaging Specification (FMS) and lower layers of the communications stack. The NI-FBUS Communications Manager hides the low-level details of Virtual Communication Relationships (VCRs), connection management, addresses, and Object Dictionary indices, and offers name access to physical devices, Virtual Field Devices (VFDs), function blocks, transducer blocks, and parameters.

The NI-FBUS Communications Manager API is independent of the National Instruments Fieldbus hardware and your operating system. With the NI-FBUS Communications Manager, you can plug multiple National Instruments Fieldbus interfaces of any type into the same PC, and use them through the NI-FBUS Communications Manager API. NI-FBUS is capable of using the AT-FBUS, PCMCIA-FBUS, and ControlNet-to-Fieldbus linking devices as its interface.

The NI-FBUS Communications Manager is interface independent because it does not require you to specify which Fieldbus interface to use in NI-FBUS Communications Manager calls. It determines over which interface to send certain Fieldbus messages. The NI-FBUS Communications Manager lets you write applications that are as independent as possible of the actual configuration of your Fieldbus interfaces.

The NI-FBUS Communications Manager API is useful for developing host applications. Typical examples are function block tuning software packages and applications for monitoring a function blocks, diagnosing a network, and developing interfaces to Human-Machine Interface (HMI) packages.

The following table lists the important elements of the NI-FBUS Communications Manager software and describes the function of each element.

Table 1-1. NI-FBUS Communication Manager Components

Component	Operating System	Description	File Name
Driver	Windows 2000/NT	A Windows kernel-mode driver that handles communication between the NI-FBUS Communications Manager and the Fieldbus interface. Installed in your standard Windows drivers directory.	`nifb.sys`
	Windows 98/95	A Windows virtual device driver that handles communication between the NI-FBUS Communications Manager and the Fieldbus interface. Installed in your Windows System directory.	`nifb.vxd`
NIFB Binary	Windows 2000/NT/ 98/95	An executable that constitutes the NIFB process. This process must be running for your NI-FBUS applications to work.	`nifb.exe`
Binary Stack File	Windows 2000/NT/ 98/95	The binary stack file that the NI-FBUS Communications Manager downloads to a Fieldbus interface. This file is a binary image of the FOUNDATION Fieldbus communication stack.	`ffstack.bin`
Dynamic Link Libraries	Windows 2000/NT	The dynamic link library installed in your Windows directory. Necessary for your application to communicate with the NIFB process.	`nifb.dll`
	Windows 98/95	A dynamic link library installed in your Windows directory. The NIFB process needs `drvintf.dll` to communicate with the VXD driver.	`drvintf.dll`
NI-FBUS Dialog utility	Windows 2000/NT/ 98/95	Lets you interact with your devices over the Fieldbus by opening descriptors, making single NI-FBUS calls, and viewing the results. Also used to verify installation and device operation, and to learn the NI-FBUS Communications Manager API.	`nifbldg.exe`

Table 1-1. NI-FBUS Communication Manager Components (Continued)

Component	Operating System	Description	File Name
NI-FBUS Interface Configuration utility	Windows 2000/NT	Lets you add Fieldbus interfaces or change the hardware configuration parameters for your Fieldbus interface. Also lets you assign a logical name to each port on the Fieldbus interface and provide the NI-FBUS Communications Manager with device description location information.	`fbconf.exe`
	Windows 98/95	Lets you assign logical names to your interface ports. In Windows 98/95, you can view hardware configuration parameters with this utility, but must use the Windows Device Manager to modify them.	`fbconf.exe`
Static Library	Windows 2000/NT/ 98/95	The static library that your application must link with in order to communicate with the NIFB process. `nifb.lib` is compiled with Microsoft C. `nifb_bor.lib` is provided for Borland C users.	`nifb.lib`

Sample source code files	Windows 2000/NT/ 98/95	Source code files that you can use with the `nifb.lib` library to make a sample application. All of these files ship with the NI-FBUS Communications Manager software, but you must use your own compiler to create the sample application.	`nifbtest.c`
			`nifb_mt.c`
			`nifbdd.c`

Sample Link Active Scheduler file	Windows 2000/NT/ 98/95	A sample Link Active Schedule file. Refer to Appendix A, *Configuring the Link Active Schedule File*, for information about configuring this file.	`sched.ini`

Table 1-1. NI-FBUS Communication Manager Components (Continued)

Component	Operating System	Description	File Name
Header files	Windows 2000/NT/ 98/95	An include file that contains some data type declarations, error code declarations, and function prototypes. In addition to `nifbus.h`, there are fourteen other header files that `nifbus.h` includes for you. Therefore, your application only has to include `nifbus.h`.	`nifbus.h`
Readme file	Windows 2000/NT/ 98/95	A documentation file that contains important information about the NI-FBUS Communications Manager software.	`readme.txt`

Installing the OPC NI-FBUS Server

From a DOS command prompt, run the following commands from the target directory:

```
regsvr32 opccomn_ps.dll
regsvr32 opcproxy.dll
nifb.exe/regserver
```

2

Developing Your Application

The NI-FBUS functions are classified in four categories:

- Administrative functions
- Core functions
- Alert and trend functions
- Device description functions

All NI-FBUS functions are described in detail in the *NI-FBUS Communications Manager Function Reference Manual.*

Administrative Functions

You can use the administrative functions to get the list of physical devices in a link, get a list of virtual field devices in a physical device, and get a list of blocks (resource, function, transducer) from a virtual field device. The administrative functions include `nifGetDeviceList`, `nifGetVfdList`, and `nifGetBlockList`. Typically, you have to call these before you call a core, alert, or any other administrative function.

Because you can use the NI-FBUS Communications Manager to communicate with each of the FOUNDATION Fieldbus entities, such as links, physical devices, virtual field devices, and blocks, there are `nifOpen` calls for you to open and get a descriptor to each of these entities.

Example: Using Administrative Functions

Suppose you want to get a descriptor to a block with `nifOpenBlock` before you read or write the block parameters. Then, you want to open a block using the block's tag.

To open a block with the tag `TI101_Analog_Input`, you invoke `nifOpenBlock(sessionDesc, "TI101_Analog_Input", &blockDesc)`, where `sessionDesc` is the descriptor of the session that you have established with the NI-FBUS Communications Manager. The NI-FBUS Communications Manager returns the descriptor of the block that you have opened in `blockDesc`. From then on, you can use this descriptor for calls associated with this block.

Core Functions

Core NI-FBUS functions are the functions that deal with processing function block parameters—primarily the `nifReadObject` and `nifWriteObject` functions, which read and write block parameters. The NI-FBUS Communications Manager encapsulates the device description services with the core function `nifGetObjectAttributes`, which gives you the device description attributes of any parameter.

Function blocks contain view or display objects. As the name implies, these objects are a collection of parameters in function blocks that are typically displayed in an operator console. Four view objects are defined for each of the ten standard function blocks in the FOUNDATION Fieldbus specification.

The following examples are a good summation of the NI-FBUS Communications Manager, because they demonstrate that details such as VCRs, indices, and connections are hidden by the *TAG.PARAMETER* access the NI-FBUS Communications Manager provides. However, to correctly write an application using the NI-FBUS Communications Manager, you must be familiar with the *Foundation Specification: Function Block Application Process Parts 1 and 2*—the standard blocks, their parameters, and their syntax—and have an idea of the architecture of Fieldbus. Refer to the *FOUNDATION Fieldbus Overview* document for an outline of Fieldbus architecture.

Example: Using Core Functions

Suppose the object `VIEW_1` for a PID function block consists of `GAIN`, `RATE`, `SP`, `CAS_IN`, `MODE`, and `ALARM_SUM` parameters of the PID function block. You want to get the values of all these parameters using a single read of the `VIEW_1` object. If the tag of a PID function block is `TIC101_PID`, you can read the `VIEW_1` object by executing the following function call:

```
nifReadObject(sessionDesc, "TIC101_PID.VIEW_1", buffer,
&cnt)
```

Notice that it is not necessary to have a block descriptor to read the object's parameters. If you *do* have the block descriptor, you can read the object with the following call:

```
nifReadObject(blockDesc, "VIEW_1", buffer, &cnt)
```

You can get the block descriptor using `nifOpenBlock`, which returns `blockDesc`.

If you wanted to change the setpoint of the preceding PID block, you could do so with the following call:

```
nifWriteObject(sessionDesc, "TIC101_PID.SP", buffer,
cnt)
```

Alert and Trend Functions

When a properly configured device detects an alarm condition, it broadcasts the data. A host device should receive the alarm and send a communication acknowledgment and an operator acknowledgment to the field device. The field device can also collect trends based on a configured sample type and interval. When it collects 16 samples, it broadcasts the trend data on the Fieldbus. Any number of interested hosts can collect this data. For more details, refer to the *Foundation Specification: Function Block Application Process Part 1*.

With a program such as the NI-FBUS Configurator, you can configure the FOUNDATION Fieldbus field devices to broadcast alert and trend data.

The NI-FBUS Communications Manager has functions to receive trends and alerts from configured devices and to perform operator acknowledgment on alerts. `nifWaitAlert` lets you wait for an alert from any device in a link, any function block in a physical device, or a specific function block, depending on the type of descriptor that you pass it. When the NI-FBUS Communications Manager receives an alert, it returns a structure containing information about the alert. The NI-FBUS Communications Manager sends the communication acknowledgment to the device automatically. The NI-FBUS Communications Manager provides a separate function, `nifAcknowledgeAlarm`, to send the operator acknowledgment.

Similarly, `nifWaitTrend` lets you wait for a trend from any device in a link, any function block in a physical device, or a specific function block, depending on the type of descriptor you pass it. When the NI-FBUS Communications Manager receives a trend, it returns a structure containing information about the trend, along with the trend data itself.

`nifWaitAlert` and `nifWaitTrend` wait until an alert or trend is received before returning, so you might want to have separate threads invoke these functions.

Device Description Functions

The NI-FBUS Communications Manager gives your applications access to device descriptions, which are binary files that describe the characteristics of blocks and parameters. Your application can use the NI-FBUS function `nifGetObjectAttributes` to decode attributes of parameters including data type, data size, help strings, and other attributes defined in the *Device Description Language Specification*. In addition, device description symbol files are used automatically to assist in allowing your applications to access parameters by name.

The NI-FBUS Communications Manager is shipped with device descriptions for all standard Fieldbus Foundation function blocks. The NI-FBUS Communications Manager can provide attributes for all parameters of all *standard* function blocks, even if the device manufacturers for your devices did not provide device descriptions. However, to get the attributes of parameters of nonstandard (not FOUNDATION Fieldbus-defined) blocks, the NI-FBUS Communications Manager requires that the device manufacturer provide the device description.

NI-FBUS supports device description menus and methods. When NI-FBUS tries to locate a device description file (`.ffo` & `.sym`) for a device, it uses the file with the latest device description revision for a given `MANUFAC_ID`, `DEV_TYPE`, and `DEV_REV`. For more information about device descriptions, refer to the *FOUNDATION Fieldbus Overview* document or your getting started manual.

Using the NI-FBUS Communications Manager Process

For any of your NI-FBUS Communications Manager applications to run correctly, you must successfully launch the NIFB process. The NIFB process is the medium by which your application communicates with the devices on the Fieldbus network. The NIFB process receives requests from your application and passes them on to the specified Fieldbus device through the Fieldbus interface connected to your machine. Refer to the *Start the NIFB Process* chapter in your getting started manual for instructions on how to start the NIFB process.

At startup, the NIFB process downloads the Fieldbus Foundation communication stack file `ffstack.bin` to the Fieldbus interfaces connected to your machine. It then downloads the communication stack configuration parameters, such as the Fieldbus network address for the

interface device, and so on, to each interface device. You can edit these parameters using the NI-FBUS Interface Configuration utility by clicking the **Advanced** button on the dialog box for the **Port** information. The advanced parameters affect the operation of the communication stack and should only be changed if you are aware of the effect of your changes on the stack.

You must make sure to specify a unique, non-default Fieldbus network address for the NI-FBUS Communications Manager to work properly. You can use a default address if another entity on the Fieldbus assigns your interface a non-default address. You can change the address from the NI-FBUS Interface Configuration utility in the **Port** dialog box. You must restart the NI-FBUS Communications Manager for any changes you make to take effect.

The NI-FBUS Communications Manager process features non-volatile storage of all network parameters, including the last known Link Active Schedule. After network parameters (including the Link Active Schedule) are stored, the NI-FBUS Communications Manager automatically reloads them to the interface on startup.

At installation time, the non-volatile copy of the schedule is empty, but you can make the NI-FBUS Communications Manager store the non-volatile Link Active Schedule by downloading it to your Fieldbus interface. To download a Link Active Schedule to your Fieldbus interface, you can use the NI-FBUS Dialog utility. Refer to Chapter 3, *NI-FBUS Dialog Utility*, for an example of how to download the Link Active Schedule to your Fieldbus interface. You can also use the NI-FBUS Configurator to download a Link Active Schedule to your Fieldbus interface.

Developing Your NI-FBUS Communications Manager Application

This section contains information to help you develop your NI-FBUS Communications Manager application.

Choose Your Level of Communication

While a few functions require a specific type of descriptor (for example, `nifGetDeviceList` requires a link descriptor), many functions (such as the core and alert and trend functions) let you communicate using any type of descriptor. With these functions, the descriptor type you choose depends on what is most convenient for you in designing your application, because

there is no significant difference in performance between the different types.

For example, if it is convenient for your application to use only a session descriptor and to keep track of tags for each block (so that you refer to all parameters in *BLOCKTAG.PARAMNAME* format), you should write your application this way. If it is easier for you to keep track of a descriptor for each block rather than a tag for each block, you should open a block descriptor for each block you are communicating with, keep track of that descriptor value, and access parameters by *PARAMNAME* using the block descriptor.

Choose to Access by Name or Index

The NI-FBUS Communications Manager supports access by name or by index for all block parameters. National Instruments recommends that you access all variables by name. Although access by index might be slightly faster in some cases, an application cannot always reliably determine indices.

The NI-FBUS Communications Manager can convert the parameter name you specify to the final index that FOUNDATION Fieldbus protocols must use to access the parameter over the network. The NI-FBUS Communications Manager converts the name to an index using standard Fieldbus Foundation-specified methods, which include a check to the device at run time to verify the index. If you hard code indices, you will have to modify them when the devices they are accessing become replaced, upgraded, or have new blocks created on them.

Choose to Write Single-Thread or Multi-Thread Application

All NI-FBUS functions are synchronous, meaning that the calling function is blocked until the NI-FBUS call completes. A Fieldbus device usually takes tens of milliseconds to respond to a block parameter read or write. It takes longer if any communication errors occur. The NI-FBUS Communications Manager uses the protocol connections to communicate with the devices. If a connection is lost, the NI-FBUS Communications Manager tries to reestablish the connection. When a connection is lost, an NI-FBUS read or write call may take several seconds to complete.

Single-Thread Applications

If potential delays like the ones discussed in the previous paragraph are acceptable for your application, you can write your application or the Fieldbus access part of your application as a single thread. Single-threaded

applications are easier to develop, debug, and test, because you do not have to consider exclusion between threads. If you are writing an application for testing, monitoring, or configuring a single device, a single-threaded application might be adequate.

Multi-Thread Applications

If your application monitors or tests several devices at a time, communication delays might affect the throughput of your application and therefore be unacceptable. If so, you can develop a multi-threaded application to improve the performance of your application. There are several ways to multi-thread your application.

If you are accessing information from function blocks or transducer blocks, you might want to create a thread for each block. Each block's thread reads and writes information from that block. If creating a thread for each block is excessive, you might consider an architecture in which you have a set of threads dedicated to Fieldbus I/O. Your application can then interface with I/O threads through a shared queue in which threads put their I/O requests. When the I/O completes, the I/O threads can inform the application by passing a message or some other synchronization scheme.

If your application performs trending or alarm handling, you might want to have separate threads that perform these functions. You can make a thread wait for a trend or alarm with the `nifWaitTrend` or `nifWaitAlert` function and then process the trend or alarm when it arrives. If you are monitoring the live list (the current list of devices on the bus), you may have a dedicated thread that calls `nifGetDeviceList`, because the call will not return until the live list changes.

Access Object Dictionary Entries

If you want to access object dictionary entries that do not reside in a block, you can access them by object dictionary index along with a virtual field device descriptor. You can access trend and linkage objects by name using a virtual field device descriptor. To access trend objects by name, either from an application program or from the NI-FBUS Dialog utility, use the name `TREND.`*`X`*, where *`X`* is a number from 1 to the number of trend objects in the virtual field device. To access linkage objects, use `LINKAGE.`*`X`*, where *`X`* is a number from 1 to the number of linkage objects in the virtual field device. If *`X`* exceeds the number of linkage objects or trend objects in the virtual field device, the NI-FBUS Communications Manager returns the `E_ORDINAL_NUM_OUT_OF_RANGE` error code.

Access Management Information Base (MIB) Parameters

To access Management Information Base parameters directly, either from a program or from the NI-FBUS Dialog utility, open the physical device you want to communicate with, and open a virtual field device on the device with the tag `MIB`. You can use the resulting virtual field device descriptor to access the MIB parameters by index or by their names (as described in the FOUNDATION Fieldbus Specification). For example, to write the macrocycle duration, access the MIB parameter `MACROCYCLE_DURATION`, and to read the live list, access the object named `LIVE_LIST_STATUS`. This method works both on local interface devices and on remote devices over the Fieldbus.

Some MIB parameters are elements of a list (such as the list of function block schedule entries or VCR entries). You can use the name for these items with a `.`*X* appended, where *X* is the element in the list you want to access. For example, the first function block schedule entry in the MIB is named `FB_START_ENTRY.1`, and the first VCR static entry in the MIB is named `VCR_STATIC_ENTRY.1`. If *X* exceeds the number of objects of that type in the MIB, the NI-FBUS Communications Manager returns the `E_ORDINAL_NUM_OUT_OF_RANGE` error code.

Because most of these parameters have to do with network configuration, a network configurator, such as the NI-FBUS Configurator, can best set these parameters.

Please keep in mind that the NI-FBUS Communications Manager manages some MIB objects internally. For instance, the NI-FBUS Communications Manager builds up internal data structures for some MIB objects, especially VCRs, etc. Manually changing the existing VCRs through a MIB descriptor could possibly lead to problems with using the NI-FBUS Communications Manager.

MIB List Parameters

```
FB_START_ENTRY
MAX_TOKEN_HOLD_TIME
SCHEDULE_DESCRIPTOR
VCR_STATIC_ENTRY
VFD_REF_ENTRY
```

MIB Parameters

AP_CLOCK_SYNC_INTERVAL
BOOT_OPERAT_FUNCTIONAL_CLASS
CHANNEL_STATES
CONFIGURED_LINK_SETTING
CURRENT_LINK_SETTING
CURRENT_TIME
DEV_ID
DLME_BASIC_CHARACTERISTICS
DLME_BASIC_INFO
DLME_LINK_MASTER_INFO
LINK_SCHEDULE_ACTIVATION
LINK_SCHEDULE_LIST_CHARACTERISTICS
LIVE_LIST_STATUS
LOCAL_TIME_DIFF
MACROCYCLE_DURATION
OPERATIONAL_POWERUP
PD_TAG
PLME_BASIC_CHARACTERISTICS
PLME_BASIC_INFO
PRIMARY_AP_TIME_PUBLISHER
PRIMARY_LINK_MASTER_FLAG
SM_SUPPORT
STACK_CAPABILITIES
T1
T2
T3
TIME_LAST_RCVD
TIME_PUBLISHER_ADDR
VCR_LIST_CHARACTERISTICS
VERSION_OF_SCHEDULE

Use the NI-FBUS Dialog Utility to Communicate with Devices

The NI-FBUS Dialog utility helps you perform simple tests of your whole Fieldbus setup, including the NI-FBUS Communications Manager, your interface board(s), and any devices you have. The NI-FBUS Dialog utility has dialog boxes that call the NI-FBUS Communications Manager API and allows you to specify parameters and make NI-FBUS calls. For example, you can use the NI-FBUS Dialog utility to get a list of devices on your network, as well as view and set parameters in each device. For more information on using the NI-FBUS Dialog utility, refer to Chapter 3, *NI-FBUS Dialog Utility*.

Write Your Application

Use the following guidelines to make sure your application uses the NI-FBUS Communications Manager interface properly.

- Always call `nifOpenSession` early in your program and check the return value of the call. This check verifies that the NI-FBUS Communications Manager process is running, which is a prerequisite for your application to access the Fieldbus network. If this call fails, your application should inform the user that the Fieldbus is currently inaccessible.
- Always close any descriptors that you open before your program exits, including session descriptors. The NI-FBUS Communications Manager requires that your application close all descriptors that it opens.
- Always check the return values from NI-FBUS calls. The NI-FBUS Communications Manager is a high-level API and performs many operations that can fail because of incorrect parameters, incorrect bus configuration, or communication failures. An application that fails to check return values might use output parameters from NI-FBUS calls that are NULL or uninitialized, leading to incorrect behavior or a program crash.
- If you plan to call any of the indefinitely-blocking functions including `nifGetDeviceList`, `nifWaitAlert`, and `nifWaitTrend`, you should probably use a separate descriptor for these calls. To terminate these calls early, you have to close the descriptor. Having a separate descriptor will ensure that terminating these calls does not affect any other NI-FBUS calls your application has pending.
- If the NI-FBUS Communications Manager stops for any reason, any outstanding calls in your application complete with the error `E_SERVER_CONNECTION_LOST`. At this point, all of the descriptors that you have (including the session) are invalid. If you restart the NI-FBUS Communications Manager, your application should recover by opening a new session to the NI-FBUS Communications Manager and opening all new descriptors. After this recovery procedure, your application should be fully operational.

Compile, Link, and Run Your Application

To compile, link, and execute your application, you must carry out the following steps:

- Add the line `#include "nifbus.h"` to any of your source files that make NI-FBUS calls. The `nifbus.h` file is located in the `includes` subdirectory of your installation. Also, make sure that the `includes` subdirectory is included in your project's settings.
- Link your application with `nifb.lib` (or `nifb_bor.lib`, if you are using Borland's compiler), which is located in the `libs` subdirectory of your installation.
- Make sure that `nifb.dll` is present in your Windows directory. `nifb.dll` is an interface DLL required to interface to the NIFB process. `nifb.dll` must be present when your application runs.
- Make sure that the NIFB process has started up and is entirely initialized before your application makes its first NI-FBUS call.
- Make sure your compiler has the structure padding or alignment parameter set to 8 bytes. This will allow proper communication of data structures.
- The `nifbus.h` header file and `nifb.lib` library have been compiled and linked with Microsoft Visual C, versions 4.0, 4.1, and 4.2. The `nifbus.h` header file and `nifb_bor.lib` library have been compiled and linked with Borland C++, version 5.0 only.

Sample Programs and Borland Compilers

The NI-FBUS Communications Manager software includes three sample programs: `nifbtest.c`, `nifb_mt.c`, and `nifbdd.c`. These files provide you with some examples of the NI-FBUS Communications Manager API usage. Notice that `nifb_mt.c` uses some Microsoft C++ specific thread creation APIs; therefore, Borland C++ users cannot use this file as it is. You must substitute these calls with the corresponding Borland C++ APIs.

Borland users should use `nifb_bor.lib` instead of `nifb.lib`. National Instruments only supports Borland C++ version 5.0 and later. Borland C++ 4.5 users must upgrade their compiler to use `nifb_bor.lib`.

Because NI-FBUS uses a device description library from the Fieldbus Foundation, the header files from the device description library are also part of the NI-FBUS include directory. Borland C++ users might have some compiler errors with these header files in multi-threaded applications.

3

NI-FBUS Dialog Utility

Note If you have the NI-FBUS Configurator, you generally will not want to use the dialog utility, since the NI-FBUS Configurator has the same functionality and is easier to use.

The NI-FBUS Dialog utility lets you interact with your devices over the Fieldbus by opening descriptors, making single NI-FBUS calls, and viewing the results. You might want to use the NI-FBUS Dialog utility to verify installation and device operation, or to learn the NI-FBUS Communications Manager API.

When you open the NI-FBUS Dialog utility, a window appears containing a single item called **Open Descriptors**. This is the root of a tree that shows an icon for each of the NI-FBUS descriptors you open using the NI-FBUS Dialog utility. The area below the icon remains empty until you make an NI-FBUS call to open a descriptor. When you open a descriptor, the NI-FBUS Dialog utility adds an icon representing that descriptor.

You can use the NI-FBUS Dialog utility to perform operations on the descriptors you have opened. Select the operation you want to perform on a descriptor by right-clicking on the descriptor icon and choosing an item on the menu that appears, or by selecting the icon with a single click and choosing an item on the **Actions** menu. The choices that appear on the menu depend on the type of descriptor you have selected.

NI-FBUS Dialog Examples

These examples describe the typical steps you go through when using the NI-FBUS Dialog utility. Before you begin the examples, open the NI-FBUS Dialog utility:

- **Windows NT 3.51**—Double-click on the **NIFBus Dialog** icon in your **NI-FBUS** program group.
- **Windows 2000/NT 4.0/98/95**—Select **Start»Programs»National Instruments FBUS»NI-FBUS Dialog**.

Example 1. Get a Device List

1. Open the NI-FBUS Dialog utility.
2. Right-click on the **Open Descriptors** icon and select **Open Session**.
3. In the **Open Session** dialog box that appears, click on the **Open Session** button. The NI-FBUS Dialog utility makes an `nifOpenSession` call to the NIFB process. This call opens a session descriptor, which represents your connection to the NIFB process.

   If the call succeeds, the NIFB process is running and responding to requests, and a new session descriptor is created under the **Open Descriptors** icon. If the call fails, make sure that the NIFB process is running and that it has not displayed any error message boxes during startup. You can check this by maximizing and looking at the `nifb.exe` console window. If the title bar does not end in "(Running)", NI-FBUS did not start up correctly.
4. Click on the **Session** icon and then select the **Actions** menu.

   The list that appears represents the NI-FBUS Communications Manager API calls you can make with a session descriptor.
5. Choose the **GetInterfaceList** function from the list of choices. This choice displays the logical name of all known interfaces.
6. Highlight the interface name of your choice and click on the **OpenLink** button.

   or

   Open a link by choosing the **OpenLink** function and entering the interface name.
7. Right-click on the **Link** icon and choose **GetDeviceList**. The NI-FBUS Dialog utility displays a list of active devices on your Fieldbus link. Your Fieldbus interface board is also included in this list.

Example 2. Download a Schedule to an Interface

1. Complete all the steps of *Example 1. Get a Device List*.
2. Select an interface board by clicking on an entry in the device list that has an asterisk (*) on its left.
3. Click on the **Open Device** button. A new dialog box appears with the identifying information for the interface board already filled in.
4. Click on the **Open Device** button on the new dialog box. If the call completes successfully, a new icon for the device descriptor appears in the tree structure on the screen.

5. Right-click on the new device icon, and select the **DownloadLASSched** menu option. A new dialog box appears with identifying information for the device already filled in.
6. In the new dialog box, click on **Browse** to locate your `.ini` file that contains the LAS schedule you want to download, or enter the full path to the file, if you know it. For more information about using the LAS with a `.ini` file, refer to Appendix A, *Configuring the Link Active Schedule File*.
7. Click on the **Load & Activate** button. The NI-FBUS Communications Manager downloads the schedule to the interface board and activates it immediately.

Example 3. Read a Parameter Using *TAG.PARAM* Access

1. Open the NI-FBUS Dialog utility.
2. Click on the **Actions** menu and select **Open Session**.
3. Click on the **Open Session** button. If the call succeeds, the NIFB process is running and responding to requests, and a new session descriptor is created under the **Open Descriptors** icon.
4. Right-click on the session descriptor icon and select **ReadObject**.
5. In the dialog box that appears, enter the name of the parameter to read in the *`BLOCKTAG.PARAM`* format, where *`BLOCKTAG`* is the tag of the block containing the parameter, and *`PARAM`* is the name of the parameter. For example, to read the `OUT` parameter of an Analog Input block called FT-201, enter `FT-201.OUT`.
6. Click on the **Read** button to perform the read operation. If the call completes successfully, the NI-FBUS Dialog utility automatically determines the type of the data and displays it in the **Data** box. If the call fails, the error message appears in the **Result** box.

Example 4. Wait for a Trend

Note You will not be able to receive any trends unless you have configured a device to generate them and configured an interface to receive them. Use your configuration software package to do this. You can also view the trend from the Dialog utility as numbers only (with no graphs).

1. Open the NI-FBUS Dialog utility.
2. Click on the **Actions** menu and select **Open Session**.

3. Click on the **Open Session** button. If the call succeeds, the NIFB process is running and responding to requests, and a new session descriptor is created under the **Open Descriptors** icon.
4. Right-click on the session descriptor icon and select **WaitTrend**.
5. The dialog box that appears waits until the NI-FBUS Communications Manager receives a trend from any device on the bus. The trend data is displayed in the **Results** box when the trend is received. The **Trend** dialog continues to wait for and display trends as they are received until you close it with the **Cancel** button.

   You can wait on trends from all types of descriptors, not just session descriptors. For example, if you wait on a trend from a device descriptor, the dialog box only displays trends coming from the device that the specified descriptor represents. The same is true of link, virtual field device, and block descriptors.

To exit the NI-FBUS Dialog utility, select **Exit** from the **File** menu.

A

Configuring the Link Active Schedule File

If you want to do scheduling and use publishers and subscribers, you must follow the instructions in this appendix; otherwise, you do not need this information. You may ignore this appendix if there is no schedule, if the schedule is downloaded over the network to your Fieldbus interface, or if you are using software such as the NI-FBUS Configurator.

Introduction to the Link Active Schedule File

You must download the Link Active Schedule file to your Fieldbus interface before the board can have Link Active Scheduler functionality on the Fieldbus network.

Save the Link Active Schedule file as a `.ini` file. You can download this file to your interface board using the NI-FBUS Dialog utility.

For detailed information about the parameters in the Link Active Schedule file, refer to the *Data Link Layer* section of the Final Specification version of the *FOUNDATION Fieldbus Specification.*

Format of the Link Active Schedule File

Create your Link Active Schedule file according to the following format.

The names of the sections of the Link Active Schedule file are:

```
[Schedule Summary]
...
[Subschedule 1]
...
[Sequence 1-1]
...
[Sequence 1-n]
...
```

```
[Subschedule x]
...
[Sequence x-1]
...
[Sequence x-y]
...
```

The general line format for all other lines is:

VARIABLE=VALUE

where the valid variable names and values are defined in Tables A-1 to A-4.

Table A-1. Valid Variable Names and Values for the Schedule Summary Section

Variable Name	Valid Values	Implied Units	Default
encodingVersionNumber	0–7	none	none
versionNumber	0x0–0xffff	none	none
builderIdentifier	0x100–0xfff	none	none
numSubSchedules	0–255	none	none
maxSchedulingOverhead	0x0–0x3f	octets	none
macroCycle	0x0–0xffffffff	1/32 ms	none

Table A-2. Valid Variable Names and Values for the Subschedule Section

Variable Name	Valid Values	Implied Units	Default
period	0x0–0xffffffff	1/32 ms	none
numSequence	0–255	none	none

Table A-3. Valid Variable Names and Values for the Sequence Section

Variable Name	Valid Values	Implied Units	Default
maxDuration	0x0–0xffff	1/32 ms	none
numElement	0–255	none	none

For the variables in Table A-4, *N* is an integer between 1 and numElement. Repeat these variables within this subschedule section exactly numElement times.

Table A-4. Valid Variable Names Including the Variable *N* and Values for the Sequence Section

Variable Name	Valid Values	Implied Units	Default
priority*N*	TIMEAVAILABLE URGENT NORMAL	none	none
address*N*	Parameter name in *TAG.PARAM* format, or DLCEP (Data Link Connection End Point) in 0x*NNNN* format	none	none

B

Troubleshooting

This appendix contains information about how to identify and solve problems with the NI-FBUS Communications Manager software.

Startup Problems

If the NIFB process is unable to find information it needs to start up, error messages will appear. You may ignore these messages and continue; however, this will result in your application not being able to communicate with the interface devices for which the error messages appeared. These messages tell you the information that the NIFB process is looking for, but cannot find.

If NI-FBUS is unable to connect to and initialize an interface device, and you decide to continue NI-FBUS startup, NI-FBUS will not try to reconnect to that interface again. This is true of all interface types supported by this software.

If an AT-FBUS or PCMCIA-FBUS interface is configured as a basic device, a link master device must be present on this link before NI-FBUS will start up.

Problems Using NI-FBUS with Windows 95

When using Windows 95, your application programs may lose contact with the NI-FBUS Communications Manager after extended use. In most cases, you can restart the application to re-establish communication with NI-FBUS.

After installation and hardware configuration, you must reboot the machine twice in order to successfully launch NI-FBUS.

Call to Open Session Fails

If the call fails, make sure that your NI-FBUS Communications Manager process is running, and that it has not displayed any error message boxes during startup. You can check this by maximizing and looking at the `nifb.exe` console window. If the title bar does not end in "(Running)," NI-FBUS did not start up correctly.

Set Address Problems

If you are having trouble setting the address of your device, you may need to change some of the System Management Info parameters in the Advanced settings of your interface port in the NI-FBUS Interface Configuration utility. The parameters involved in setting addresses are T1 and T3. These parameters represent delay time values that your interface card uses to compensate for the delays inherent in the device and in the set address protocol itself.

T1 is a parameter that describes the expected response delay of the device at a given address. Normally, you will not need to increase this parameter; however, if it appears that your interface card is not seeing the device's responses related to setting addresses, you can increase this value. The correct value for this parameter can be dependent on the number of devices on the link. For example, if you are using a bus monitor, you might see a `WHO_HAS_PD_TAG` request going to the device to start the Set Address sequence, and an `IDENTIFY` response coming back, but with the host never continuing on to the next step of the protocol (the `SET_ADDRESS` packet). This would probably mean that your T1 value is too small and should be increased.

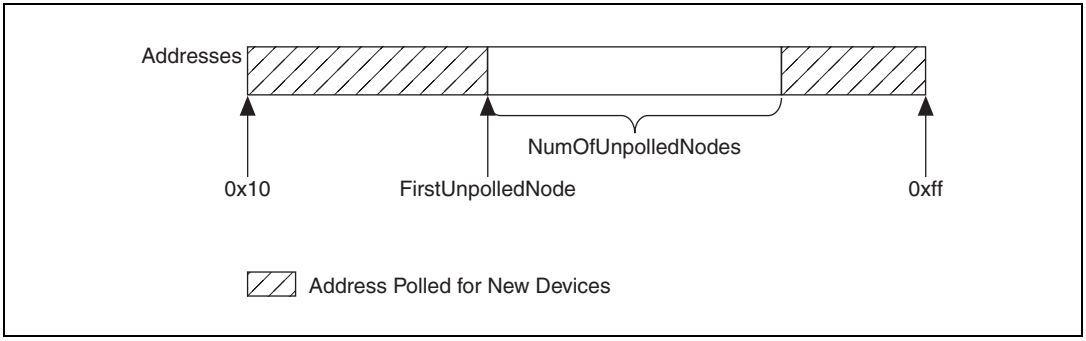
T3 is a parameter that describes the expected time for the device to respond at its new address. This parameter is highly dependent on the number of devices on the link, and the number of addresses being polled (see *Setting Number of Polled Addresses* later in this appendix for instructions on how to set the number of polled addresses). If you are using a bus monitor, you may be able to see the host identify a device (with the `IDENTIFY` packet) at the new address, before the devices has sent its probe response (PR) packet to the host. This is an error that is indicative of a T3 value that is too small; if this occurs, increase your T3 value until the `IDENTIFY` to the new address occurs after the PR.

All of the System Management Info timers are in units of 1/32 of a millisecond; for instance, T3=32000 units means that T3=1 second.

Setting Number of Polled Addresses

The Fieldbus specification describes how a Link Active Scheduler device (LAS device) probes a list of addresses to allow devices to come online during normal operation. The LAS sends a Probe Node (PN on the bus monitor) packet to each address in its list of addresses every so often during operation, where “every so often” is a time that depends on the number of devices on the link, and the setting of the Link Maintenance Token Hold Time parameter.

The Fieldbus specification describes how to tell the LAS to “skip” probing certain addresses in the range to speed up how long it takes to detect new devices on the bus (or devices that are having their addresses changed). The two parameters involved in maintaining the list are called `FirstUnpolledNode` and `NumOfUnpolledNodes`, and they can be found in the NI-FBUS Interface Configuration utility advanced settings for a port, in the DLME Master Info section. The following diagram shows how the LAS determines the list:



In other words, FirstUnpolledNode tells the LAS the beginning of a region of addresses to not probe, and NumOfUnpolledNodes tells the LAS the length of that region. So if FirstUnpolledNode were 0x25, and NumOfUnpolledNodes were 0xba, then no addresses from 0x25 to 0xdf would be probed. That means that if a device with an address of 0x25 were placed on this bus, the LAS would not probe it, and it would never be able to send or receive packets on the bus.

The reason to have a NumOfUnpolledNodes whose value is nonzero is as follows. The LAS probes every address in the list, then starts over again at the beginning. Because a device cannot come on the bus until its address is probed, if the LAS is probing all 255–16+1=240 possible addresses, and each probe node request goes out every T milliseconds, it might take 240T milliseconds for a device to get on the bus. If, however, the LAS

probed only the first 16 addresses and the last 16 addresses, it might take 32T milliseconds for the device to get on the bus; this results in the new device being recognized almost 8 times faster.

These parameters also affect the Set Address protocol, because recognizing a device at a new address is really the same as recognizing a completely new device, as the new address must be probed for the device to come online. In this way, the NumOfUnpolledNodes parameter can affect the value of the Set Address protocol parameter "T3", which is described in the section above. For example, increasing the NumOfUnpolledNodes parameter might fix a SetAddress T3 problem because it takes the device less time to be recognized at the new address.

Applications-Related Information

Because of a clarification in the FOUNDATION Fieldbus Specification AR (Anomaly Report) #22, the order of bits read from bit string type parameters has been reversed. This will be the case in the National Instruments Device Developer Kit release 2.2 and for devices from other manufacturers after this AR. This may impact host applications that read or write to bit string parameters (such as the MODE_BLK parameter).

Note that this is not an NI-FBUS change, but a device implementation issue that may affect your application running on NI-FBUS.

Using Fieldbus with OPC

Starting with version 2.3.5, NI-FBUS is an OPC server. This OPC server can configure and use multi-variable containers (if supported by the Fieldbus device) to access multiple tags in a device concurrently. This server is compliant with version 1.0a of the OPC specifications with some added support for FOUNDATION Fieldbus bitstring types and array elements.

An OPC client utility is provided with the NI-FBUS software to let you browse Fieldbus OPC tags. Follow the instructions listed in the *Installing the OPC NI-FBUS Server* section in Chapter 1, *Introduction*, to get the OPC server operational.

NI-FBUS support two modes of operation for OPC: traditional polled mode, and high-speed multi-variable container mode. Multi-variable container mode typically improves performance by ten to twenty times over traditional polling. The multi-variable container mode is based

on the preliminary multi-variable container specification of the Fieldbus Foundation.

The FP-3000 firmware revision 2.3.5 is recognized by NI-FBUS as being multi-variable container capable. Use the following instructions to use multi-variable container mode with the FP-3000 and your OPC software package.

Lookout

1. Create one or more OPCClient objects in your Lookout process.
2. Select the OPCNifbus server from the drop-down list of OPC servers.
3. Set the Activate member of the OPC client object to FALSE using one of the following methods:
   - Edit the connections for the OPCClient object and set the Activate member to FALSE.
   - Create a switch object on the Control Panel with the position source set to Remote. Then, set the Remote source to the Activate member of the OPCClient object. Leave edit mode, then set the switch to the off (FALSE) position.
4. Add all the items you are interested in to the OPCClient object(s).
5. Set the Activate member of the OPC client object to TRUE using one of the following methods:
   - Edit the connections for the OPCClient object and set the Activate member to TRUE.
   - Set the switch to the on (TRUE) position.

A similar deactivation/activation procedure will have to be followed while opening a previously saved `.lkp` process file. The Lookout process will always go live immediately when it is loaded. The OPC client object's Activate member is always set to TRUE at startup (even though the switch position may indicate off/FALSE).

To get the maximum read throughput from the FP-3000 after your saved Lookout process starts up, follow these steps:

1. Set the Activate members of the OPCClient object to FALSE. This deactivates the OPC group.
2. Set the Activate member to TRUE. This activates the OPC group, the therefore the creation and use of multi-variable container objects by the OPC server.

Server Explorer

1. Launch the Server Explorer.
2. Create an inactive OPC client group:
   a. Right-click on OPCNifbus and select **Add/Edit Groups**.
   b. Create a group with the appropriate parameters. Make sure you uncheck the **Active** box.
3. Add all items.
4. Select **File»OPC»Save** to save the file.
5. Activate the group by right-clicking on the group and selecting **Activate Group**.
6. When you open the file you saved and want to go live, right-click on **OPCNifbus** and select **Connect to Server**. After Server Explorer has connected to the server, activate the group as described in Step 5.

BridgeVIEW

Stop (but do not quit) the BridgeVIEW engine before you add any items to your current configuration. Allow the engine between two and five minutes to shut down, especially if your tag configuration file has a large number of items. When you are done adding items, restart the BridgeVIEW engine.

Problems Using OPC

If you download a new configuration (from the same station or another station), the multi-variable container objects on the FP-3000 may be destroyed. If this happens, the OPC server defaults to reading views on the FP-3000 and the throughput will be reduced. The severity of this reduction depends on factors such as the number of points polled and the group update period. To force the OPC server to recreate and reuse multi-variable container objects, you must deactivate the client OPC group, the reactivate it. The procedure for doing this varies depending on the software you are using. This procedure is defined in the preceding sections for Lookout and BridgeVIEW.

If you change or clear the address of the FP-3000, you must deactivate then reactivate the client OPC group to force the OPC server to recreate and reuse the multi-variable container objects.

Since multi-variable container object transmissions are unscheduled, you should leave at least 30% of your macrocycle free for unscheduled traffic.

Problems Using Fieldbus with Lookout

Fieldbus Objects do not Appear in Lookout

If you want to use the native Fieldbus objects in Lookout, you have to delete the lookout.dat file in the Lookout directory. This file is an index file that tells Lookout what objects it has available. Fieldbus objects are not available by default. Lookout will regenerate the lookout.dat file the next time it is started. When it regenerates the file, it will see that Fieldbus software has been installed and will make the Fieldbus objects available.

Fieldbus Alarms in Lookout

In Lookout, there is a separate alarms window for Fieldbus alarms. Under the Options menu, choose Fieldbus to show this window. The window can also be shown using traditional Lookout datamember ShowAlarms. See the entry for National Instruments Fieldbus in the Lookout object reference manual (also available from the Help menu within Lookout.

If you want alarms to appear in the main alarm window (rather than the Fieldbus alarms window), you need to create Lookout alarm objects.

C

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `www.ni.com`

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `www.ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `www.ni.com`

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `www.ni.com`. Branch office web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}

A

Address	Character code that identifies a specific location (or series of locations) in memory.
Administrative Function	An NI-FBUS function that deals with administrative tasks, such as returning descriptors and closing descriptors.
Alarm	A notification the NI-FBUS Communications Manager software sends when it detects that a block leaves or returns to a particular state.
Alarm condition	A notification that a Fieldbus device sends to another Fieldbus device or interface when it leaves or returns to a particular state.
Alert	An alarm or event.
Alert function	A function that receives or acknowledges an alert.
Analog	A description of a continuously variable signal or a circuit or device designed to handle such signals.
API	See Application Programmer Interface.
Application	Function blocks.
Application Programmer Interface	A message format that an application uses to communicate with another entity that provides services to it.
Array	Ordered, indexed list of data elements of the same type.

B

Basic device	A device that can communicate on the Fieldbus, but cannot become the LAS.
Bitstring	A data type in the object description.
Block	A logical software unit that makes up one named copy of a block and the associated parameters its block type specifies. The values of the parameters persist from one invocation of the block to the next. It can be a resource block, transducer block, or function block residing within a virtual field device.
Block tag	A character string name that uniquely identifies a block on a Fieldbus network.
BridgeVIEW	A program for developing applications that require high channel count datalogging, as well as supervisory control of distributed systems.
Buffer	Temporary storage for acquired or generated data.
Bus	The group of conductors that interconnect individual circuitry in a computer. Typically, a bus is the expansion vehicle to which I/O or other devices are connected. Examples of PC busses are the ISA and PCI buses.

C

Channel	A pin or wire lead to which you apply or from which you read the analog or digital signal.
Client	A device that sends a request for communication on the bus.
Communication stack	Performs the services required to interface the user application to the physical layer.
Connection Management	The service the NI-FBUS Communications Manager provides by handling Virtual Communication Relationships.
ControlNet	A 5 Mbit/sec communications protocol based on Producer/Consumer technology.
Core Function	The basic functions that the NI-FBUS Communications Manager software performs, such as reading and writing block parameters.

D

Data Link Layer	The second-lowest layer in the ISO seven-layer model (layer two). The Data Link Layer splits data into frames to send on the physical layer, receives acknowledgment frames, and re-transmits frames if they are not received correctly. It also performs error checking to maintain a sound virtual channel to the next layer.
Descriptor	A number returned to the application by the NI-FBUS Communications Manager, used to specify a target for future NI-FBUS calls.
Device	A sensor, actuator, or control equipment attached to the Fieldbus.
Device Description	A machine-readable description of all the blocks and block parameters of a device.
Device Description Language	A formal programming language that defines the parameters of the blocks. It also defines attributes of parameters and blocks like help strings in different languages, ranges of values for parameters, and so on.
Device Description Service	A set of functions that applications use to access Device Descriptions.
Directory	A structure for organizing files into convenient groups. A directory is like an address showing where files are located. A directory can contain files or subdirectories of files.
DLL	See Dynamic Link Library.
Driver	Device driver software installed within the operating system.
Dynamic Link Library	A library of functions and subroutines that links to an application at run time.

F

FB	Function Block.
Field device	A Fieldbus device connected directly to a Fieldbus.
Fieldbus	An all-digital, two-way communication system that connects control systems to instrumentation. A process control local area network defined by ISA standard S50.02.
Fieldbus Foundation	An organization that developed a Fieldbus network specifically based upon the work and principles of the ISA/IEC standards committees.

Fieldbus Messaging Specification	The layer of the communication stack that defines a model for applications to interact over the Fieldbus. The services FMS provides allow you to read and write information about the OD, read and write the data variables described in the OD, and perform other activities such as uploading/downloading data and invoking programs inside a device.
Fieldbus Network Address	Location of a board or device on the Fieldbus; the Fieldbus node address.
FMS	See Fieldbus Messaging Specification.
FOUNDATION Fieldbus specification	The communications network specification that the Fieldbus Foundation created.
FP-3000	National Instruments network interface module for the FieldPoint I/O system.
Function block	A named block consisting of one or more input, output, and contained parameters. The block performs some control function as its algorithm. Function blocks are the core components you control a system with. The Fieldbus Foundation defines standard sets of function blocks. There are ten function blocks for the most basic control and I/O functions. Manufacturers can define their own function blocks.
Function Block Application	The block diagram that represents your control strategy.

H

Hard code	To permanently establish something that should be variable in a program.
Header file	A C-language source file containing important definitions and function prototypes.
HMI	Human-Machine Interface. A graphical user interface for the process with supervisory control and data acquisition capability.
Host device	A computer or controller on a Fieldbus network.

I

I/O	Input/output.
Index	An integer that the Fieldbus specification assigns to a Fieldbus object or a device that you can use to refer to the object. A value in the object dictionary used to refer to a single object.

K

Kbits	Kilobits.
Kernel	The set of programs in an operating system that implements basic system functions.
Kernel mode	The mode in which device drivers run on Windows NT.

L

LAS	See Link Active Scheduler.
Link	A FOUNDATION Fieldbus network is made up of devices connected by a serial bus. This serial bus is called a link (also known as a segment).
Link Active Schedule	A schedule of times in the macrocycle when devices must publish their output values on the Fieldbus.
Link Active Scheduler	The Fieldbus device that is currently controlling access to the Fieldbus. A device that is responsible for keeping a link operational. The LAS executes the link schedule, circulates tokens, distributes time, and probes for new devices.
Link master device	A device that is capable of becoming the LAS
Linkage	A connection between function blocks.
Linkage object	An object resident in a device that defines connections between function block input and output across the network. Linkage objects also specify trending connections.
Live list	The list of all devices that are properly responding to the Pass Token.

LM	Link Master.
Lookout	National Instruments Lookout is a full-featured object-based automation software system that delivers unparalleled power and ease of use in demanding industrial measurement and automation applications.

M

Macrocycle	The least common multiple of all the loop times on a given link, or one iteration of a the process control loop.
Menu	An area accessible from the command bar that displays a subset of the possible command choices. In the NI-FBUS Configurator, refers to menus defined by the manufacturer for a given block.
Method	Methods describe operating procedures to guide a user through a sequence of actions.
Mode	Type of communication.

N

Network address	The Fieldbus network address of a device.
Network Management	A layer of the FOUNDATION Fieldbus communication stack that contains objects that other layers of the communication stack use, such as Data Link, FAS, and FMS. You can read and write SM and NM objects over the Fieldbus using FMS Read and FMS Write services.
Nifb.exe	The NIFB process that must be running in the background for you to use your AT-FBUS or PCMCIA-FBUS interface to communicate between the board and the Fieldbus.
NI-FBUS API	The NI-FBUS Communications Manager.
NI-FBUS Communications Manager	Software shipped with National Instruments Fieldbus interfaces that lets you read and write values. It does not include configuration capabilities.
NI-FBUS Configurator	National Instruments Fieldbus configuration software. With it, you can set device addresses, clear devices, change modes, and read and write to the devices.
NI-FBUS Fieldbus Configuration System	See NI-FBUS Configurator.

NI-FBUS process	Process that must be running in the background for you to use your AT-FBUS or PCMCIA-FBUS interface to communicate between the board and the Fieldbus.
Node	Junction or branch point in a circuit.

O

Object	An element of an object dictionary.
Object Dictionary	A structure in a device that describes data that can be communicated on the Fieldbus. The object dictionary is a lookup table that gives information such as data type and units about a value that can be read from or written to a device.
Octet	A single 8-bit value.
OPC	OLE for Process Control.
Output parameter	A block parameter that sends data to another block.

P

Parameter	One of a set of network-visible values that makes up a function block.
PC	Personal Computer.
PCMCIA	Personal Computer Memory Card International Association.
PD	Proportional Derivative.
Physical device	A single device residing at a unique address on the Fieldbus.
PID	Proportional/Integral/Derivative. A common control function block algorithm that uses proportions, integrals, and derivatives in calculation.
PN	Probe Node.
Poll	To repeatedly inspect a variable or function block to acquire data.
Port	A communications connection on a computer or remote controller.
PR	Probe Response.

Program	A set of instructions the computer can follow, usually in a binary file format, such as a `.exe` file.
Publisher	A device that has at least one function block with its output value connected to the input of another device.

S

s	Seconds.
Sample type	Specifies how trends are sampled on a device, whether by averaging data or by instantaneous sampling.
Segment	See Link.
Server	Device that receives a message request.
Service	Services allow user applications to send messages to each other across the Fieldbus using a standard set of message formats.
Session	A communication path between an application and the NI-FBUS Communications Manager.
Stack	A set of hardware registers or a reserved amount of memory used for calculations or to keep track of internal operations.
Static library	A library of functions/subroutines that you must link to your application as one of the final steps of compilation, as opposed to a Dynamic Link Library, which links to your application at run time.
Stub	See Spur.
Subscriber	A device that has at least one function block with its input value connected to the output of another device.
Surge	Large, unwanted voltage or current on wires. Generally caused by lightning or nearby heavy electrical power use.
Surge suppressor	A device used to discharge surges to ground.
Symbol file	A Fieldbus Foundation or device manufacturer-supplied file that contains the ASCII names for all the objects in a device.
System Management	A layer of the FOUNDATION Fieldbus communication stack that assigns addresses and physical device tags, maintains the function block schedule for the function blocks in that device, and distributes application time. You can also locate a device or a function block tag through SM.

T

Tag	A name you can define for a block, virtual field device, or device.
Thread	An operating system object that consists of a flow of control within a process. In some operating systems, a single process can have multiple threads, each of which can access the same data space within the process. However, each thread has its own stack and all threads can execute concurrently with one another (either on multiple processors, or by time-sharing a single processor).
Transducer block	A block that is an interface to the physical, sensing hardware in the device. It also performs the digitizing, filtering, and scaling conversions needed to present input data to function blocks, and converts output data from function blocks. Transducer blocks decouple the function blocks from the hardware details of a given device, allowing generic indication of function block input and output. Manufacturers can define their own transducer blocks.
Trend	A Fieldbus object that allows a device to sample a process variable periodically, then transmit a history of the values on the network.
Trend function	An NI-FBUS call related to trends.

V

VCR	See Virtual Communication Relationship.
VFD	See Virtual Field Device.
View objects	Predefined groupings of parameter sets that HMI applications use.
Virtual Communication Relationship	Preconfigured or negotiated connections between virtual field devices on a network.
Virtual Field Device	The virtual field device is a model for remotely viewing data described in the object dictionary. The services provided by the Fieldbus Messaging Specification allow you to read and write information about the object dictionary, read and write the data variables described in the object dictionary, and perform other activities such as uploading/downloading data and invoking programs inside a device. A model for remotely viewing data described in the object dictionary.

W

Waveform	Multiple voltage readings taken at a specific sampling rate.

Index

L

M

N

O

P

R

S

T

W