Fieldbus

NI-FBUS™ Communications Manager Function Reference Manual

NATIONAL INSTRUMENTS™

July 2000 Edition
Part Number 321288C-01

Important Information

Conventions

The following conventions are used in this manual:

	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.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`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

Tables

1 Administrative Functions

For details on how NI-FBUS functions are classified and how to use them, refer to the *NI-FBUS Communications Manager User Manual*.

Related Documentation

- *Function Block Application Process, Part 1*
- *Function Block Application Process, Part 2*
- *Device Description Services Specification, Fieldbus Foundation*
- *Fieldbus Message Specification, Fieldbus Foundation*

List of Administrative Functions

Table 1-1. List of Administrative Functions

Function	Purpose
`nifClose`	Close an open descriptor
`nifDownloadDomain`	Download data to the virtual field device (VFD) domain
`nifGetBlockList`	Return a list of information for all blocks of the specified type present in the VFD
`nifGetDeviceList`	Return the list of information for all active devices on the network
`nifGetInterfaceList`	Read the list of interface names from the NI-FBUS Communications Manager configuration
`nifGetVFDList`	Gather VFD information on a specified physical device
`nifOpenBlock`	Return a descriptor representing a block
`nifOpenLink`	Return a descriptor representing a Fieldbus link
`nifOpenPhysicalDevice`	Return a descriptor representing a physical device
`nifOpenSession`	Return a descriptor for an NI-FBUS session
`nifOpenVfd`	Return a descriptor representing a VFD

nifClose

Purpose

Close an open descriptor.

Format

```
nifError_t nifClose(nifDesc_t ud)
```

Input

`ud`	The descriptor from an `nifOpen` call.

Output

Not applicable.

Context

Block, VFD, physical device, link, session.

Description

`nifClose` closes the specified descriptor. The descriptor is invalid after it is closed. Be sure your application closes all the descriptors it opens. Your application should always close a descriptor if it no longer needs the descriptor.

If you close a descriptor with calls pending on it, the calls complete within the usual time with an error code indicating that you closed the descriptor prematurely. If you make more synchronous wait calls that wait on the closing descriptor, such as `nifWaitTrend`, `nifWaitAlert`, and `nifGetDeviceList`, the NI-FBUS Communications Manager aborts these functions and returns an error code indicating that you closed the descriptor. Because calls that wait on a closed descriptor return an error message, you should have a separate descriptor just for these synchronous wait calls.

Note A *session* is a connection between your application and an NI-FBUS entity. If you close a session, you close the communication channel between your application and the NI-FBUS entity associated with the session. Make sure you close all descriptors opened under this session before closing a session descriptor.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor is invalid.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifDownloadDomain

Purpose

Download data from `fileName` to the specified VFD domain according to the `index` value.

Format

```
nifError_t nifDownloadDomain (nifDesc_t ud, uint16 index, char
                              *fileName)
```

Input

`ud`	The descriptor of the VFD you are accessing by `index`.
`index`	The absolute VFD index value of the domain you specified to download the data.
`fileName`	The name of the file where the download data is stored.

Context

VFD, physical device, link, session.

Description

`nifDownloadDomain` is used to download the data or parameter values to the specified VFD domain. The domain is specified by `index`.

To determine the index value you need, consult the documentation of the device to which you are trying to download the domain. If the device supports the Domain Download feature, the index for download should be specified in the documentation.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor you specified is not valid.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communication Manager, under which the descriptor was opened, has been lost or closed.
`E_RESOURCE`	The NI-FBUS Communications Manager is unable to allocate some system resource; this is usually a memory problem.
`E_DEVICE_CHANGED`	The device you specified is changed.
`E_VFD_CHANGED`	The VFD you specified is changed.

nifGetBlockList

Purpose

Returns a list of information for all blocks of the specified type present in the VFD.

Format

```
nifError_t nifGetBlockList(nifDesc_t ud, uint8 whichTypes,
               nifBlockInfo_t *info, uint16 *numBlocks)
```

Input

`ud`	The descriptor of a VFD.
`whichTypes`	Specifies what types of blocks to return (function, transducer, or physical).
`numBlocks`	The number of buffers allocated in the `info` list.

Output

`info`	The list of information associated with each block.
`numBlocks`	The number of blocks actually in the VFD.

Context

VFD.

Description

`nifGetBlockList` returns information about all the blocks in the specified VFD. A *`block`* can be a resource block, transducer block, or function block residing within a VFD. Only blocks of the types specified by `whichTypes` are returned.

To determine how many list items are to be returned in the call, call the function twice. The first time you call the function, set the `numBlocks` parameter to 0. The function will return an error stating that there were not enough buffers configured, and it will return a new number for `numBlocks`. Use this new `numBlocks` parameter to allocate memory for the data. When you call the function the second time, use this new parameter. By doing so, you will allocate only as much memory as necessary.

`nifBlockInfo_t` is defined as follows:

```
typedef struct {
   char             fbTag[TAG_SIZE + 1];
   uint16           startIndex;
   uint32           ddName;
   uint32           ddItem;
```

```
    uint16          ddRev;
    uint16          profile;
    uint16          profileRev;
    uint32          executionTime;
    uint32          periodExecution;
    uint16          numParams;
    uint16          nextFb;
    uint16          startViewIndex;
    uint8           numView3;
    uint8           numView4;
    uint16          ordNum;
    uint8           blockType;
} nifBlockInfo_t;
```

The `blockType` field in `nifBlockInfo_t` can be `FUNCTION_BLOCK`, `TRANSDUCER_BLOCK`, or `RESOURCE_BLOCK`.

The `whichTypes` parameter must be a bit combination of `FUNCTION_BLOCK`, `TRANSDUCER_BLOCK`, and `RESOURCE_BLOCK`.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor was invalid or of the wrong type.
`E_COMM_ERROR`	The NI-FBUS Communications Manager failed to communicate with the device.
`E_BUF_TOO_SMALL`	The buffer does not contain enough entries to hold all the information for the blocks. If you receive this error, buffer entries that you allocated do not contain valid block information when the call returns.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifGetBlockList` completed.
`E_BAD_ARGUMENT`	The `whichtypes` value is something other than `FUNCTION_BLOCK`, `TRANSDUCER_BLOCK`, or `RESOURCE_BLOCK`.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage.
`E_BAD_DEVICE_DATA`	The device returned some inconsistent information.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifGetDeviceList

Purpose

Return the list of information for all active devices on the network.

Format

```
nifError_t nifGetDeviceList(nifDesc_t link,
             nifDeviceInfo_t *devInfo, uint16 *numDevices,
             uint16 *revision)
```

Input

`link`	The link descriptor to return information for.
`numDevices`	The number of allocated list entries.
`revision`	The revision number from the last `nifGetDeviceList` call, or zero (see the *Description* for usage).

Output

`devInfo`	The list of device information.
`numDevices`	The number of devices present in the link.
`revision`	Current revision number of the live list that the NI-FBUS Communications Manager reads from the Fieldbus interface to the specified link.

Context

Link.

Description

`nifGetDeviceList` returns a list of information describing each device on the link. A *link* is a group of Fieldbus devices connected across a single wire pair with no intervening bridges. Before `nifGetDeviceList` returns the list of information, `nifGetDeviceList` waits until the revision argument passed in differs from the live list revision number the Fieldbus interface keeps to the specified link. The revision numbers the Fieldbus interface keeps start at one, so if you pass in a zero for `revision`, you can force `nifGetDeviceList` to immediately return the current device list. To use `nifGetDeviceList` most effectively, in subsequent calls to it, you should pass in the `revision` parameter output from the previous call to `nifGetDeviceList`. Using the revision parameter output from the previous call forces `nifGetDeviceList` to wait until the device list has actually changed before returning the list of information.

If a device on the bus is unresponsive, its entry in the device information list has the tag and device ID `unknown device`, but its address field is correct. Also, the flag bit `NIF_DEV_NO_RESPONSE` is set.

The device list includes devices in the fixed, temporary, and visitor address ranges.

If there are too few input buffers, `nifGetDeviceList` returns an error code, but the `numDevices` parameter is set to the total number of devices available. In this case, the buffers you pass in do *not* contain valid data, but the revision number is set to the correct value. If a device is an interface device, then the flag bit `NIF_DEV_INTERFACE` is set. You can abort a pending `nifGetDeviceList` call by closing the link descriptor on which the call was made.

To determine how many list items are to be returned in the call, call the function twice. The first time you call the function, set the `numDevices` parameter to 0. The function will return an error stating that there were not enough buffers configured, and it will return a new number for `numDevices`. Use this new `numDevices` parameter to allocate memory for the data. When you call the function the second time, use this new parameter. By doing so, you will allocate only as much memory as necessary.

`nifDeviceInfo_t` is defined as follows:

```
typedef struct {
   char deviceID[DEV_ID_SIZE + 1];
   char pdTag[TAG_SIZE + 1];
   uint8 nodeAddress;
   uint32 flags;
} nifDeviceInfo_t;
```

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The link descriptor is invalid.
`E_BUF_TOO_SMALL`	There are not enough buffers allocated. If you receive this error, your input buffers do not contain valid data.
`E_COMM_ERROR`	The NI-FBUS Communications Manager failed to communicate with the device.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifGetDeviceList` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifGetInterfaceList

Purpose

Read the list of interface names from the NI-FBUS Communications Manager configuration.

Format

```
nifError_t nifGetInterfaceList(nifDesc_t ud,
           int16 *numIntf, nifInterfaceInfo_t *info)
```

Input

`ud`	A valid session descriptor.
`numIntf`	The number of buffers for interface information reserved in `info`.

Output

`numIntf`	The actual number of names returned.
`info`	An array of structures containing the interface name and device ID for each interface.

Context

Not applicable.

Description

`nifGetInterfaceList` returns the interface name and device ID of each Fieldbus interface in the NI-FBUS Communications Manager configuration. The `numIntf` parameter is an IN/OUT parameter. On input, it must contain the number of buffers that info allocates and points to, and on output it contains the total number of interface information entries available. If not enough buffers were allocated, or if the `info` buffer is NULL, the NI-FBUS Communications Manager returns an error and does not copy any data to the buffers. In this case, the `numIntf` parameter is still valid.

To determine how many list items are to be returned in the call, call the function twice. The first time you call the function, set the `numIntf` parameter to 0. The function will return an error stating that there were not enough buffers configured, and it will return a new number for `numIntf`. Use this new `numIntf` parameter to allocate memory for the data. When you call the function the second time, use this new parameter. By doing so, you will allocate only as much memory as necessary.

The `nifInterfaceInfo_t` structure is defined as follows:

```
typedef struct nifInterfaceInfo_t{
   char      interfaceName[NIF_NAME_LEN];
   char      deviceID[DEV_ID_SIZE +1];
} nifInterfaceInfo_t;
```

Note `nifGetInterfaceList` is an internal function for the NI-FBUS Communications Manager and does not cause Fieldbus activity.

Return Values

`E_OK`	The call was successful.
`E_BUF_TOO_SMALL`	The buffer does not contain enough entries to hold all the interface information.
`E_CONFIG_ERROR`	Some configuration information, such as registry information or network configuration information, is incorrect.

nifGetVFDList

Purpose

Gather VFD information on a specified physical device.

Format

```
nifError_t nifGetVFDList(nifDesc_t ud, nifVFDInfo_t *info,
             uint16 *numBuffers)
```

Input

`ud`	The descriptor of the physical device to get the VFD list for.
`numBuffers`	The number of buffers allocated in the `info` list.

Output

`numBuffers`	The number of VFDs actually in the device.
`info`	The VFD information.

Context

Physical device.

Description

`nifGetVFDList` gathers function block application VFD information from the specified physical device.

If there are too few input buffers, or if the input buffer pointer is NULL, an error code is returned, but the `numBuffers` parameter is set to the total number of VFDs in the device. In this case, no buffers contain valid data on output.

To determine how many list items are to be returned in the call, call the function twice. The first time you call the function, set the `numBuffers` parameter to 0. The function will return an error stating that there were not enough buffers configured, and it will return a new number for `numBuffers`. Use this new `numBuffers` parameter to allocate memory for the data. When you call the function the second time, use this new parameter. By doing so, you will allocate only as much memory as necessary.

The `info` parameter has the following format:

```
typedef struct {
   char      vfdTag[TAG_SIZE + 1];
   char      vendor[TAG_SIZE +1];
   char      model[TAG_SIZE +1];
   char      revision[TAG_SIZE +1];
```

```
    int16     ODVersion;
    uint16    numTransducerBlocks;
    uint16    numFunctionBlocks;
    uint16    numActionObjects;
    uint16    numLinkObjects;
    uint16    numAlertObjects;
    uint16    numTrendObjects;
    uint16      numDomainObjects;
    uint16      totalObjects;
    uint32      flags;
} nifVFDInfo_t;
```

Return Values

`E_OK`	The call was successful.
`E_COMM_ERROR`	The NI-FBUS Communications Manager failed to communicate with the device.
`E_INVALID_DESCRIPTOR`	The input descriptor does not correspond to a physical device.
`E_BUF_TOO_SMALL`	There were not enough allocated buffers. Your specified input buffers do *not* contain valid data.
`E_SM_NOT_OPERATIONAL`	The device is present, but cannot respond because it is at a default address.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifGetVFDList` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.
`E_BAD_DEVICE_DATA`	The device returned some inconsistent information.

nifOpenBlock

Purpose

Return a descriptor representing a block.

Format

```
nifError_t nifOpenBlock (nifDesc_t ud, char *blockTag,
             nifDesc_t *out_ud)
```

```
nifError_t nifOpenBlock (nifDesc_t ud, NIFB_ORDINAL(n),
             nifDesc_t *out_ud)
```

Input

`ud`	A valid session, link, physical device, or VFD descriptor.
`blockTag`	The tag of the block. To access a block by ordinal number within a VFD, use the `NIFB_ORDINAL` macro in the `nifbus.h` header file. You can only access a block by ordinal number for VFD descriptors.

Output

`out_ud`	A descriptor for the block you request.

Context

VFD, physical device, link, session.

Description

`nifOpenBlock` returns a descriptor for the block you specify. You must pass a valid session, link, physical device, or VFD descriptor to this function.

There are two ways to specify the block: by tag, and by ordinal number. To open the block by its tag, you must set `blockTag` to the current tag of the block. The NI-FBUS Communications Manager returns an error if it finds more than one block with the same tag. You can obtain the list of block tags within a specified VFD with a call to `nifGetBlockList`.

To open the block by its ordinal number, use the `NIFB_ORDINAL` macro. This macro is only valid if `ud` is a VFD descriptor. The first block in a VFD has the ordinal number zero. Notice that the first block in a VFD is always the resource block.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The input descriptor is invalid.
`E_MULTIPLE`	There are identical block tags.
`E_ORDINAL_NUM_OUT_OF_RANGE`	The ordinal number is out of the device's range.
`E_COMM_ERROR`	An error occurred when the NI-FBUS Communications Manager communicated with the device.
`E_NOT_FOUND`	There is no such block in the device or VFD with the specified tag.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifOpenBlock` completed.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.
`E_BAD_DEVICE_DATA`	The device returned some inconsistent information.

nifOpenLink

Purpose

Return a descriptor representing a Fieldbus link.

Format

```
nifError_t nifOpenLink (nifDesc_t session, uint8 interfaceOrDevID,
             char *name, uint16 linkID, nifDesc_t *out_ud)
```

Input

`session`	A valid session descriptor on which to open the link.
`interfaceOrDevID`	How to specify the link: zero if by interface name, one if by local device ID.
`name`	The interface name or local device ID.
`linkID`	The link ID.

Output

`out_ud`	A descriptor for the link you request.

Context

Session.

Description

`nifOpenLink` returns a descriptor for the link you specify. You must pass a valid session descriptor to this function.

There are two ways you can specify the link. If the `interfaceOrDevID` parameter is zero, then `name` specifies the name of the interface the link is connected to. The list of valid interface names is contained in a configuration source which the NI-FBUS Communications Manager has access to, and can be obtained by a call to `nifGetInterfaceList`. If `interfaceOrDevID` is one, then the `name` specifies the device ID of an interface device to which the NI-FBUS Communications Manager is attached.

In both cases, `linkID` is the Fieldbus link ID number for the specified link. For single-link Fieldbus networks, you can set `linkID` to zero.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The input descriptor is invalid.
`E_CONFIG_ERROR`	Some configuration information, such as registry information or network configuration information, is incorrect.
`E_NOT_FOUND`	The interface name, device ID, or link ID you specified is not found.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage.
`E_BAD_ARGUMENT`	The `interfaceOrDevID` value is not valid.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifOpenLink` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifOpenPhysicalDevice

Purpose

Return a descriptor representing a physical device.

Format

```
nifError_t nifOpenPhysicalDevice (nifDesc_t ud, uint8 tagOrDevID,
             char *name, nifDesc_t *out_ud)
```

Input

`ud`	A valid session or link descriptor on which to open the device.
`tagOrDevID`	How to specify the device: zero if by physical device tag, one if by device ID.
`name`	The tag or device ID.

Output

`out_ud`	A descriptor for the device you request

Context

Link, session.

Description

`nifOpenPhysicalDevice` returns a descriptor for the physical device you specify. You must pass a valid session or link descriptor to this function. If you pass a link descriptor, the NI-FBUS Communications Manager searches only that link for the specified device.

There are two ways you can specify the device. If the `tagOrDevID` parameter is zero, then the `name` specifies the tag of the physical device. If `tagOrDevID` is one, then `name` is the device ID of the device you specify. You can obtain the list of physical device tags and device IDs of devices on the network with a call to `nifGetDeviceList`.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The input descriptor is invalid.
`E_BAD_ARGUMENT`	The `tagOrDevID` value is not valid.
`E_NOT_FOUND`	No attached physical device has the specified device ID or physical device tag.
`E_MULTIPLE`	There is more than one device with the same tag or device ID on the same Fieldbus network.

`E_COMM_ERROR`	An error occurred when the NI-FBUS Communications Manager communicated with the device.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifOpenPhysicalDevice` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifOpenSession

Purpose

Return a descriptor for an NI-FBUS Communications Manager session.

Format

```
nifError_t nifOpenSession (void *reserved, nifDesc_t *out_ud)
```

Input

`reserved`	Reserved for future use; you must set this value to NULL.

Output

`out_ud`	A descriptor for the NI-FBUS Communications Manager communications entity you request.

Context

Not applicable.

Description

`nifOpenSession` returns a descriptor for the NI-FBUS Communications Manager session. When you open a session, the NI-FBUS Communications Manager establishes a communication channel between your application and the NI-FBUS entity. All subsequent descriptors you open are associated with this session, and all the NI-FBUS calls on these descriptors communicate with the NI-FBUS entity through the communication channel established during the `nifOpenSession` call.

The `reserved` argument is reserved for future use; you must set `reserved` to NULL.

Return Values

`E_OK`	The call was successful.
`E_SERVER_NOT_RESPONDING`	Either the NI-FBUS Communications Manager server has not been started, or the server, in its current state, cannot respond to the request.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage, or a failure of file access functions.

nifOpenVfd

Purpose

Return a descriptor representing a Virtual Field Device (VFD).

Format

```
nifError_t nifOpenVfd (nifDesc_t ud, char *vfdTag,
              nifDesc_t *out_ud)

nifError_t nifOpenVfd (nifDesc_t ud, NIFB_ORDINAL(n),
              nifDesc_t *out_ud)
```

Input

`ud`	A valid physical device descriptor.
`vfdTag`	The tag of the VFD. To access by ordinal number within a physical device, use the `ORDINAL` macro in the `nifbus.h` header file.

Output

`out_ud`	A descriptor for the VFD you request

Context

Physical device.

Description

`nifOpenVfd` returns a descriptor for the VFD you specify. More than one VFD can reside within a physical device. You must pass a valid physical device descriptor to this function.

There are two ways to specify the VFD: by tag, and by ordinal number. To open the VFD by its tag, you must set the `vfdTag` parameter to the current tag of the VFD. The NI-FBUS Communications Manager returns an error if it finds more than one VFD with the same tag. You can obtain the list of VFD tags within a specified physical device with a call to `nifGetVFDList`.

To open the VFD by its ordinal number, use the `NIFB_ORDINAL` macro. The first VFD of your application in a physical device has the ordinal number zero. Notice that the Management VFDs are not included in the ordinal numbering scheme.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The input descriptor is invalid.
`E_MULTIPLE`	There are identical VFD tags.

`E_ORDINAL_NUM_OUT_OF_RANGE`	The ordinal number is out of the device's range.
`E_COMM_ERROR`	An error occurred when the NI-FBUS Communications Manager communicated with the device.
`E_NOT_FOUND`	No VFD in the device has the specified VFD tag.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage.
`E_SM_NOT_OPERATIONAL`	The device is present, but cannot respond because it is at a default address.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifOpenVfd` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.
`E_BAD_DEVICE_DATA`	The device returned some inconsistent information.

2 Core Fieldbus Functions

You can use the NI-FBUS core functions to access Fieldbus block parameters using any type of descriptor. Because there are several ways to identify the Fieldbus block parameters, the NI-FBUS core functions accept special interface macros for the `name` argument, as well as the standard *`TAG.PARAM`* identifier format. Refer to the *Using Interface Macros* section at the end of this chapter for tips on using the interface macros.

List of Core Functions

Table 2-1. List of Core Functions

Function	Purpose
`nifFreeObjectAttributes`	Free an `nifAttributes_t` structure allocated during a previous call to `nifGetObjectAttributes`.
`nifFreeObjectType`	Free an `nifObjTypeLinst_t` structure allocated during a previous call to `nifGetObjectType`.
`nifGetObjectAttributes`	Read a single set of object attributes from the Device Description (DD).
`nifGetObjectName`	Returns the Object Dictionary symbol name of the specified object.
`nifGetObjectSize`	Return the size in bytes of an object's value.
`nifGetObjectType`	Returns the Object Dictionary type of the specified object.
`nifReadObject`	Read an object's value from a device.
`nifReadObjectList`	Read the values of several objects from a device or several devices.
`nifWriteObject`	Write a parameter value to a device.

nifFreeObjectAttributes

Purpose

Free an `nifAttributes_t` structure allocated during a previous call to `nifGetObjectAttributes`.

Format

```
nifError_t nifFreeObjectAttributes(nifAttributes_t *attr)
```

Input

`attr`	Object attribute values your application reads using `nifGetObjectAttributes`.

Output

Not applicable.

Context

Session, block, VFD, physical device, link.

Description

`nifFreeObjectAttributes` frees up the memory associated with the `nifAttributes_t` structure specified by `attr.attr` must have been filled in by a successful call to `nifGetObjectAttributes`. Once this function has been called, the contents of `attr` are no longer valid.

If your application does not call this function after calling `nifGetObjectAttributes`, your application will not free up memory properly.

Return Values

`E_OK`	The call was successful.
`E_BAD_ARGUMENT`	`attr` was not a valid `nifAttributes_t` structure.

nifFreeObjectType

Purpose

Frees the `nifObjTypeList_t` structure allocated during a previous call to `nifGetObjectType`.

Format

```
nifError_t nifFreeObjectType(nifObjTypeList_t *typeData)
```

Input

`typeData`	Object Type values to be freed. These values were previously read with the `nifGetObjectType` function call.

Output

Not applicable.

Context

Session, block, VFD, physical device, link.

Description

`nifFreeObjectType` frees up the memory associated with the `nifObjTypeList_t` structure specified by `typeData`. `typeData` must have been filled in by a successful call to `nifGetObjectType`. Once this function has been called, the contents of `typeData` are no longer valid.

If your application does not call this function after calling `nifGetObjectType`, your application will not free up memory properly.

Refer to `nifGetObjectType` to get more details about the `nifObjTypeList_t` structure.

Return Values

`E_OK`	The call was successful.
`E_BAD_ARGUMENT`	`typeData` was not a valid `nifObjTypeList_t` structure.

nifGetObjectAttributes

Purpose

Read a single set of object attributes from the Device Description (DD).

Format

```
nifError_t nifGetObjectAttributes(nifDesc_t ud, char *name,
             nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_INDEX(uint16 idx), nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_INDEX_SUBINDEX(uint16 idx, uint16 subidx),
             nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_ITEM(uint32 item), nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_ITEM_SUBINDEX(uint32 item, uint16 subidx),
             nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_BLOCK_ITEM(char *blocktag, uint32 item),
             nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_BLOCK_ITEM_SUBINDEX(char *blocktag, uint32 item,
             uint16 subidx), nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_BLOCK_INDEX(char *blocktag, uint16 idx),
             nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_BLOCK_INDEX_SUBINDEX(char *blocktag, uint16 idx,
             uint16 subidx), nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_NAME_SUBINDEX(char *name, uint16 subidx),
             nifAttributes_t *attr)
nifError_t nifGetObjectAttributes(nifDesc_t ud,
             NIFB_BLOCK_NAME_SUBINDEX(char *blocktag, char *name,
             uint16 subidx), nifAttributes_t *attr)
```

Input

`ud`	The descriptor (of any type if by name; VFD or block if by index).
`name`	Name of the object you need the device description attributes of, in *`BLOCKTAG.PARAM`* form. To specify a structure element by name, specify the name in *`BLOCKTAG.STRUCT.ELEMENT`* format. Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the object.

Output

`attr`	Object attribute values read from the DDOD (Device Description Object Dictionary). The type `nifAttributes_t` consists of a data structure including a type code which selects from a list of structures, one for each type of object. Other information, including whether individual attributes were successfully evaluated and whether individual attributes are dynamic (meaning they could change) is also provided. The structure is too long to be included in this manual, so you can find it in the NI-FBUS Communications Manager header files.

Context

Session, block, VFD, physical device, link.

Description

The NI-FBUS Communications Manager reads the device description object attributes identified in the call from the DDOD associated with `ud` and returned in `attr`. Notice that the object attributes describe certain characteristics of the object, but do not contain the object's value. The device description object attributes also differ in content from the FMS Object Description of the object.

For block, VFD, physical device, or link descriptors, the object name may refer to a variable or a variable list. You would normally use `nifGetObjectAttributes` to read the type description of a certain data type.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the object.

For more detailed information concerning the `nifAttributes_t` structure, refer to the *Fieldbus Foundation Device Description Services User Guide*, Chapter 3, *Using ddi_get_item*.

Note After a successful call to `nifGetObjectAttributes`, your application must call `nifFreeObjectAttributes` when it is done using the `attr` structure. Your application will not free up memory correctly if it does not perform this operation.

Return Values

`E_OK`	The call was successful.
`E_CONFIG_ERROR`	Some configuration information, such as registry information or network configuration information, is incorrect.
`E_INVALID_DESCRIPTOR`	The device descriptor does not correspond to a VFD or block.
`E_SYMBOL_FILE_NOT_FOUND`	The NI-FBUS Communications Manager could not find the symbol file.
`E_SM_NOT_OPERATIONAL`	The device is present, but cannot respond because it is at a default address.
`E_NOT_FOUND`	The referred object does not exist, or it does not have object attributes.
`E_MULTIPLE`	The NI-FBUS Communications Manager found more than one identical tag; the function failed.
`E_ORDINAL_NUM_OUT_OF_RANGE`	The ordinal number is out of the device's range.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifGetObjectAttributes` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifGetObjectName

Purpose

Returns the Object Dictionary symbol name of the specified object.

Format

```
nifError_t nifGetObjectName(nifDesc_t ud, char *inName, char
                          *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_INDEX(uint16 idx),
                          char *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_INDEX_SUBINDEX(uint16
                          idx, uint16 subidx), char *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_ITEM(uint32 item),
                          char *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_ITEM_SUBINDEX(uint32
                          item, uint16 subidx), char *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_BLOCK_INDEX(char
                          *blocktag, uint32 idx), char *outName)

nifError_t nifGetObjectName(nifDesc_t ud,
                          NIFB_BLOCK_INDEX_SUBINDEX(char *blocktag,
                          uint16 idx, uint16 subidx), char *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_NAME_SUBINDEX(char
                          *name, uint16 subidx), char *outName)

nifError_t nifGetObjectName(nifDesc_t ud, NIFB_BLOCK_NAME_SUBINDEX
                          (char *blocktag, char *name, uint16 subidx),
                          char *outName)
```

Input

`ud`	The descriptor of the session, link, physical device, VFD or block if you are accessing by name. If you are accessing by index, `ud` must be a VFD or block.
`inName`	The name of the parameter you want to read the OD symbol name in *`BLOCKTAG.PARAM`* form. Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter. To specify a named structure element, supply name in *`BLOCKTAG.STRUCT.ELEMENT`* format.

Output

`outName`	The Object symbol name read from the Object Dictionary in the device.

Context

Session, block, VFD, DDOD, physical device, link.

Description

`nifGetObjectName` is used to read the Object Dictionary symbol names of objects such as block, VFD, MIB objects, or communication objects from devices.

- If `ud` is the descriptor of a link, then `inName` must be in *`BLOCKTAG.PARAM_NAME`* format.
- If `ud` is a session descriptor, then all links are searched for the given *`BLOCKTAG.PARAM_NAME`*. The call fails if identical *`BLOCKTAG.PARAM_NAME`* tags are found on the bus. Index access is not allowed for session descriptors.
- If `ud` is the descriptor of a general function block application VFD, and you use the `NIFB_INDEX` macro, the index specified is the index of the object in the VFD.
- If `ud` is the descriptor of a function block, name must be in *`PARAM_NAME`* format.
- If `ud` is the descriptor of a function block, and you use the `NIFB_INDEX` or `NIFB_INDEX_SUBINDEX` macro, the index specified is the relative index of the parameter within the block. Relative indices start at one for the first parameter. Index zero retrieves the object dictionary symbol name of the block itself.
- In all cases, you can expand *`PARAM_NAME`* to *`STRUCT.ELEMENT`* format to represent a named element of a named structure.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor you specified is not valid.
`E_NOT_FOUND`	The NI-FBUS Communication Manager could not find the specified object.
`E_SYMBOL_FILE_NOT_FOUND`	The NI-FBUS Communication Manager could not find the symbol file.
`E_BAD_ARGUMENT`	The object specified by index was that of a simple data type, which must already be known to you.

`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communication Manager, under which the descriptor was opened, has been lost or closed.
`E_DEVICE_CHANGED`	The device you specified is changed.
`E_VFD_CHANGED`	The VFD you specified is changed.
`E_COMM_ERROR`	An error occurred when the NI-FBUS Communication Manager tried to communicate with the device.
`E_RESOURCE`	The NI-FBUS Communications Manager is unable to allocate some system resource; this is usually a memory problem.
`E_OBSOLETE_BLOCK`	The block you specified is no longer valid.

nifGetObjectSize

Purpose

Return the size (in bytes) of an object's value.

Format

```
nifError_t nifGetObjectSize(nifDesc_t ud, char *name,
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud, NIFB_INDEX(uint16 idx),
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_INDEX_SUBINDEX(uint16 idx, uint16 subidx),
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_ITEM(uint32 item), int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_ITEM_SUBINDEX(uint32 item, uint16 subidx),
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_BLOCK_ITEM(char *blocktag, uint32 item),
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_BLOCK_ITEM_SUBINDEX(char *blocktag, uint32 item,
             uint16 subidx), int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_BLOCK_INDEX(char *blocktag, uint16 idx),
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_BLOCK_INDEX_SUBINDEX(char *blocktag, uint16 idx,
             uint16 subidx), int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_NAME_SUBINDEX(char *name, uint16 subidx),
             int16 *size_in_bytes)
nifError_t nifGetObjectSize(nifDesc_t ud,
             NIFB_BLOCK_NAME_SUBINDEX(char *blocktag, char *name,
             uint16 subidx), int16 *size_in_bytes)
```

Input

`ud`	The descriptor of a block.
`name`	Character string name of the object you need the size of, in *`BLOCKTAG.PARAM`* form. To specify a structure element by name, specify the name in *`BLOCKTAG.STRUCT.ELEMENT`* format. Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the character string name.

Output

`size_in_bytes`	The size of the object.

Context

Session, block, VFD, physical device, link.

Description

This function returns the size of the specified Object Value. You have to pass a buffer of the returned size to `nifReadObject` to hold the value of the object.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the character string name.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The specified descriptor is invalid.
`E_SYMBOL_FILE_NOT_FOUND`	The NI-FBUS Communications Manager could not find the symbol file.
`E_NOT_FOUND`	The named object does not exist.
`E_MULTIPLE`	Multiple identical tags were found; the function failed.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifGetObjectSize` completed.
`E_ORDINAL_NUM_OUT_OF_RANGE`	The ordinal number is out of the device's range.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifGetObjectType

Purpose

Returns the Object Dictionary type of the specified object.

Format

```
nifError_t nifGetObjectType(nifDesc_t ud, char *objName,
            nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_INDEX(uint16 idx), nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_INDEX_SUBINDEX(uint16 idx, uint16 subidx),
            nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_ITEM(uint32 item), nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_ITEM_SUBINDEX(uint32 item, uint16 subidx),
            nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_BLOCK_ITEM(char *blocktag, uint32 item),
            nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_BLOCK_ITEM_SUBINDEX(char *blocktag, uint32 item,
            uint16 subidx), nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_BLOCK_INDEX(char *blocktag, uint16 idx),
            nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_BLOCK_INDEX_SUBINDEX(char *blocktag, uint16 idx,
            uint16 subidx), nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_NAME_SUBINDEX(char *name, uint16 subidx),
            nifObjTypeList_t *typeData)

nifError_t nifGetObjectType(nifDesc_t ud,
            NIFB_BLOCK_NAME_SUBINDEX(char *blocktag, char *name,
            uint16 subidx), nifObjTypeList_t *typeData)
```

Input

`ud`	The descriptor of the session, link, physical device, VFD or block if you are accessing by name. If you are accessing by index, `ud` must be a VFD or block.
`objName`	The name of the parameter you want to read the OD type of, in *`BLOCKTAG.PARAM`* form. Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter. To specify a named structure element, supply `name` in *`BLOCKTAG.STRUCT.ELEMENT`* format. To specify a type index returned by a previous call to `nifGetObjectType`, use the `NIFB_TYPE_INDEX` macro.

Output

`typeData`	Object Type value read from the object dictionary in the device. The `nifObjTypeList_t` data structure is a record consisting of an object type code, the number of elements, the `blocktag` to which this object belongs (if applicable), and a pointer to a list of elements of type `nifObjElem_t`. The `nifObjElem_t` type is a structure which consists of two elements: the OD `typeIndex` of the element, and the OD length of the element.

Context

Session, block, VFD, DDOD, physical device, link.

Description

`nifGetObjectType` is used to read the Object Dictionary type values of objects such as block parameters, MIB objects, or communication parameters from devices.

- If `ud` is the descriptor of a link, then `objName` must be in *`BLOCKTAG.PARAM_NAME`* format.
- If `ud` is a session descriptor, then all links are searched for the given *`BLOCKTAG.PARAM_NAME`*. The call fails if identical *`BLOCKTAG.PARAM_NAME`* tags are found on the bus. Index access is not allowed for session descriptors.
- If `ud` is the descriptor of a general function block application VFD, and you use the `NIFB_INDEX` macro, the index specified is the index of the object in the VFD.
- If `ud` is the descriptor of a function block, `name` must be in *`PARAM_NAME`* format.
- If `ud` is the descriptor of a function block, and you use the `NIFB_INDEX` or `NIFB_INDEX_SUBINDEX` macro, the index specified is the relative index of the parameter within the block. Relative indices start at one for the first parameter. Index zero retrieves the OD type of the block itself.
- In all cases, you can expand *`PARAM_NAME to STRUCT.ELEMENT`* format to represent a named element of a named structure.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter.

The `nifObjTypeList_t` data structure is defined as follows:

```
typedef struct {
   uint8          objectCode;
   uint16         numElems;
   char           blockTag[TAG_SIZE + 1];
   nifObjElem_t   *allElems;
   } nifObjTypeList_t;
```

The `nifObjElem_t` data type is defined as follows:

```
typedef struct {
   uint16         objTypeIndex;
   uint16         objSize;
   } nifObjElem_t;
```

The `objectCode` returned in the data structure `nifObjTypeList_t` is as specified in the *FMS Specifications* in the *Fieldbus Foundation Specifications*, and is listed in Table 2-2 for your convenience.

Table 2-2. Object Codes for the `nifObjTypeList_t` Data Structure

Object	**Object Code in** `fbtypes.h`
`Domain`	`ODT_DOMAIN`
`Program Invocation`	`ODT_PI`
`Event`	`ODT_EVENT`
`Data Type`	`ODT_SIMPLETYPE`
`Data Type Structure Description`	`ODT_STRUCTTYPE`
`Simple Variable`	`ODT_SIMPLEVAR`
`Array`	`ODT_ARRAY`
`Record`	`ODT_RECORD`
`Variable List`	`ODT_VARLIST`

For object codes ODT_STRUCTTYPE, ODT_SIMPLEVAR, ODT_ARRAY, and ODT_RECORD, the list of elements in allElements contains the typeIndex and the size of each component element. For example, the following fragment of pseudocode gets the type information for a structured object and does something with the type information for each element:

```
nifObjTypeList_t typeInfo;
nifDesc_t aiBlock;
int loop;
...
nifGetObjectType(aiBlock, "OUT", &typeInfo);
for (loop=0; loop < typeInfo.numElems; loop++)
{
   doSomethingWithElement(typeInfo.allElems[loop]);
}
```

For variable list objects (type ODT_VARLIST), you must call nifGetObjectType for each element in the list of elements with the typeIndex of the element returned in the list with the first nifGetObjectType call. The typeIndex of the element returned in the list in this case is the relative index of the element within the block, whose name is returned by blockTag. These subsequent calls to nifGetObjectType should use the NIFB_INDEX macro to specify the typeIndex returned by the first call.

For example, the following fragment of pseudocode gets the type information for a variable list object and does something with the type information for each variable:

```
nifObjTypeList_t typeInfo, varTypeInfo;
nifDesc_t aiBlock;
int loop;

...

nifGetObjectType(aiBlock, "VIEW_1", &typeInfo);
if (typeinfo.objectCode == ODT_VARLIST)
{
   for (loop=0; loop < typeInfo.numElems; loop++)
   {
      nifGetObjectType(aiBlock,
         NIFB_INDEX(typeInfo.allElems[loop].objTypeIndex),
         &varTypeInfo);
      doSomethingWithVariable(varTypeInfo);
   }
}
```

For all successful calls to nifGetObjectType, you must call nifFreeObjectType to clean up memory allocated within these structures.

For objects with the object codes `ODT_DOMAIN`, `ODT_PI`, `ODT_EVENT`, and `ODT_SIMPLETYPE`, only the object type is returned, and the list of elements `allElems` in the structure `nifObjTypeList_t` is empty. The list of standard data types for an object which has the object code `ODT_SIMPLETYPE` is also as specified in the *FMS Specifications* in the *Fieldbus Foundation Specifications* and is listed in Table 2-3 for your convenience.

Table 2-3. Object Codes for the `nifObjTypeList_t` Data Structure

Data Type	`objTypeIndex` **in** `fbtypes.h`	Number of Octets (Size)
`Boolean`	`FF_BOOLEAN`	1
`Integer8`	`FF_INTEGER8`	1
`Integer16`	`FF_INTEGER16`	2
`Integer32`	`FF_INTEGER32`	4
`Unsigned8`	`FF_UNSIGNED8`	1
`Unsigned16`	`FF_UNSIGNED16`	2
`Unsigned32`	`FF_UNSIGNED32`	4
`Floating Point`	`FF_FLOAT`	4
`Visible String`	`FF_VISIBLE_STRING`	1, 2, 3, ...
`Octet String`	`FF_OCTET_STRING`	1, 2, 3, ...
`Date`	`FF_DATE`	7
`Time of Day`	`FF_TIMEOFDAY`	4 or 6
`Time Difference`	`FF_TIME_DIFF`	4 or 6
`Bit String`	`FF_BIT_STRING`	1, 2, 3, ...
`Time Value`	`FF_TIME_VALUE`	8

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor you specified is not valid.
`E_TIMEOUT`	The device containing the object is present but did not respond within the timeout period.
`E_MULTIPLE`	More than one identical tag was found; the function failed.

`E_NOT_FOUND`	The NI-FBUS Communications Manager could not find the specified object.
`E_BAD_ARGUMENT`	The object specified by index was that of a simple data type, which must already be known to you.
`E_RESOURCES`	The NI-FBUS Communications Manager is unable to allocate some system resource; this is usually a memory problem.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager, under which the descriptor was opened, has been lost or closed.

nifReadObject

Purpose

Read an object's value from a device.

Format

```
nifError_t nifReadObject(nifDesc_t ud, char *name, void *buffer,
              uint8 *length)
nifError_t nifReadObject(nifDesc_t ud, NIFB_INDEX(uint16 idx),
              void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_INDEX_SUBINDEX(uint16 idx, uint16 subidx),
void *buffer, uint8 *length)
              nifError_t nifReadObject(nifDesc_t ud,
NIFB_ITEM(uint32 item), void *buffer, uint8 *length)
              nifError_t nifReadObject(nifDesc_t ud,
NIFB_ITEM_SUBINDEX(uint32 item, uint16 subidx),
              void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_BLOCK_ITEM(char *blocktag, uint32 item),
              void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_BLOCK_ITEM_SUBINDEX(char *blocktag, uint32 item,
              uint16 subidx), void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_BLOCK_INDEX(char *blocktag, uint16 idx),
              void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_BLOCK_INDEX_SUBINDEX(char *blocktag, uint16 idx,
              uint16 subidx),void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_NAME_SUBINDEX(char *name, uint16 subidx),
              void *buffer, uint8 *length)
nifError_t nifReadObject(nifDesc_t ud,
              NIFB_BLOCK_NAME_SUBINDEX(char *blocktag, char *name,
              uint16 subidx), void *buffer, uint8 *length)
```

Input

`ud`	The descriptor of the session, link, physical device, VFD or block if reading by name. If reading by index, `ud` must be a VFD or block.
`name`	Name of the parameter your application reads, in *`BLOCKTAG.PARAM`* format. To specify a structure element by name, specify the name in `BLOCKTAG.STRUCT.ELEMENT` format. Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter.
`length`	The size of the buffer to hold the result, in bytes.

Output

`buffer`	The value that the NI-FBUS Communications Manager reads.
`length`	The actual size of the resulting data, in bytes.

Context

Session, block, VFD, physical device, link.

Description

`nifReadObject` reads the values of objects such as block parameters or communications parameters from devices.

- If `ud` is the descriptor of a link, then `name` must be in the format *`BLOCKTAG.PARAM_NAME`*.
- If `ud` is a session descriptor, then all links are searched for the given *`BLOCKTAG.PARAM_NAME`*. The call fails if multiple identical *`BLOCKTAG.PARAM_NAME`* tags are located on the bus. Index access is not allowed for session descriptors.
- If `ud` is the descriptor of a general function block application VFD, then name must be in the format *`BLOCKTAG.PARAM_NAME`*.
- If `ud` is the descriptor of a function block, `name` must be in the format *`PARAM_NAME`*.
- If `ud` is the descriptor of a function block, and the `NIFB_INDEX` or `NIFB_INDEX_SUBINDEX` macro is used, the index specified is the relative index of the parameter within the block. Relative indices start at 1 for the first parameter.
- In all descriptor cases, you can expand *`PARAM_NAME`* itself to *`STRUCT.ELEMENT`* format to represent a named element of a named structure.

In each case, `name` can represent either a variable or a variable list object. You should determine the size of the object beforehand, possibly with a call to `nifGetObjectSize`. If the object is larger than the buffer size specified in `length`, the NI-FBUS Communications Manager returns an error, and none of the data in the buffer is valid.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter.

The data `nifReadObject` returns is in Fieldbus Foundation FMS Application format. You must accomplish conversion of the data to the internal format of your processor and compiler.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor does not correspond to a VFD or function block; this descriptor is no longer valid.
`E_NOT_FOUND`	The referred object does not exist.
`E_OBJECT_ACCESS_DENIED`	The NI-FBUS Communications Manager interface does not have the required privileges. The access group you belong to is not allowed to acknowledge the event, or the password you used is wrong.
`E_MULTIPLE`	The NI-FBUS Communications Manager found more than one identical tag; the function failed.
`E_BUF_TOO_SMALL`	The object is larger than your buffer.
`E_SM_NOT_OPERATIONAL`	The device is present, but cannot respond because it is at a default address.
`E_SYMBOL_FILE_NOT_FOUND`	The NI-FBUS Communications Manager could not find the symbol file.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifReadObject` completed.
`E_COMM_ERROR`	The NI-FBUS Communications Manager failed to communicate with the device.
`E_PARAMETER_CHECK`	The device reported a violation of parameter-specific checks.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifReadObjectList

Purpose

Read the values of several objects from a device or several devices.

Format

```
nifError_t nifReadObjectList (nifDesc_t ud, char **blkParamList,
            uint16 numObjects, void *buffer, uint16 *length,
            nifError_t *errArray)
```

Input

`ud`	The descriptor of the session, link, physical device, VFD, or block.
`blkParamList`	The list of parameter names your application reads in the form of *`BLOCKTAG.PARAM`*. To specify any parameter by index use the `NIFB_INDEX` macro. To specify any parameter that is an array or structure element by index and subindex, use the `NIFB_INDEX_SUBINDEX` macro. To specify a named structure element, supply the parameter name in the form of *`BLOCKTAG.STRUCT.ELEMENT`*.
`numObjects`	The number of parameter names specified in `blkParamList`. (The maximum number of objects that can be specified in `blkParamList` is given by the constant `MAX_LIST_ELEMS`.)
`length`	The size of the buffer to hold the result of all the parameter reads, in bytes.

Output

`buffer`	The values of all the parameters read, stored as a continuous string of bytes.
`length`	The cumulative size of the actual resulting data in bytes.
`errArray`	The error codes resulting from each parameter read. The error codes have a one-to-one correspondence with the order in which the parameters are specified in `blkParamList`.

Context

Session, link, device, VFD, block.

Description

`nifReadObjectList` reads the values of objects specified in the list, which may include block parameters or communication parameters from devices.

- If `ud` is the descriptor of a link, each name in `blkParamList` must be in the format *`BLOCKTAG.PARAM_NAME`*.
- If `ud` is a session descriptor, then all links are searched for any given name specified by the `blocktag.param` format in `blkParamList`. The read of this particular object fails if identical *`BLOCKTAG.PARAM_NAME`* tags are located on the bus. Index access is not allowed for session descriptors.
- If `ud` is the descriptor of a general function block application VFD, any name in `blkParamList` must be in the format `blocktag.param_name`.
- If `ud` is the descriptor of a function block, any name in `blkParamList` must be in the format *`PARAM_NAME`*.
- If `ud` is the descriptor of a function block and the `NIFB_INDEX` or `NIFB_INDEX_SUBINDEX` macro is used to specify a name in `blkParamList`, the index specified is the relative index of the parameter within the block. Relative indices start at 1 for the first block parameter.
- In all descriptor cases, any `PARAM_NAME` specified in `blkParamList` can be expanded to *`STRUCT.ELEMENT`* format to represent a named element of a named structure.

For each name specified in `blkParamList`, the name can either represent a variable or a variable list object. You should determine the size of each object specified in `blkParamList` beforehand, possibly with a call to `nifGetObjectSize`. If the cumulative size of all the objects specified in the list is larger than the buffer size specified in `length`, the NI-FBUS Communications Manager returns an error. The data in the buffer is valid for however many objects were successfully read. The success or failure of the read for every object specified in `blkParamList` is indicated in `errArray`, the array in which error codes are returned. The error code in the first element of `errArray` is the error code indicating success or failure upon read of the first object specified in `blkParamList`, and so on.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameters in `blkParamList`.

The data `nifReadObjectList` returns is in Fieldbus Foundation FMS Application format. You must accomplish conversion of the data to the internal format of your processor and compiler.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor is no longer valid.
`E_BUF_TOO_SMALL`	The size of the data resulting from the read of all objects specified in the list is larger than your buffer.
`E_RESOURCES`	A system resource problem occurred. The resource problem is usually a memory shortage.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifWriteObject

Purpose

Write a parameter value to a device.

Format

```
nifError_t nifWriteObject(nifDesc_t ud, char *name, void *buffer,
              uint8 length)
nifError_t nifWriteObject(nifDesc_t ud, NIFB_INDEX(uint16 idx),
              void *buffer, uint8 length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_INDEX_SUBINDEX(uint16 idx, uint16 subidx),
              void *buffer, uint8 length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_ITEM(uint32 item), void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_ITEM_SUBINDEX(uint32 item, uint16 subidx),
              void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_BLOCK_ITEM(char *blocktag, uint32 item),
              void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_BLOCK_ITEM_SUBINDEX(char *blocktag, uint32 item,
              uint16 subidx), void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_BLOCK_INDEX(char *blocktag, uint16 idx),
              void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_BLOCK_INDEX_SUBINDEX(char *blocktag, uint16 idx,
              uint16 subidx), void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_NAME_SUBINDEX(char *name, uint16 subidx),
              void *buffer, uint8 *length)
nifError_t nifWriteObject(nifDesc_t ud,
              NIFB_BLOCK_NAME_SUBINDEX(char *blocktag, char *name,
              uint16 subidx), void *buffer, uint8 *length)
```

Input

`ud`	The descriptor of the session, link, physical device, VFD, or block, if writing by name. If writing by index, `ud` must be a VFD or block.
`name`	Name of the parameter you want the NI-FBUS Communications Manager to write, in *`BLOCKTAG.PARAM`* form. To specify a structure element by name, specify the name in *`BLOCKTAG.STRUCT.ELEMENT`* format. Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter.
`buffer`	The value you want the NI-FBUS Communications Manager to write.
`length`	The size of the data buffer, in bytes.

Output

Not applicable.

Context

Block, VFD, physical device, link, session.

Description

`nifWriteObject` writes the values of a function block parameter to a device.

- If `ud` is the descriptor of a session or link, then `name` must be in the format *`BLOCKTAG.PARAM_NAME`*.
- If `ud` is a session descriptor, then all links are searched for the given *`BLOCKTAG.PARAM_NAME`*. The function fails if more than one identical *`BLOCKTAG.PARAM_NAME`* match is found.
- If `ud` is a physical device descriptor, a parameter is written by *`BLOCKTAG.PARAM_NAME`*.
- If `ud` is the descriptor of a general Virtual Field Device, then name must be in the format *`BLOCKTAG.PARAM_NAME`*.
- If `ud` is the descriptor of a function block, `name` must be in the format *`PARAM_NAME`*.
- If `ud` is the descriptor of a function block, and you use the `NIFB_INDEX` or `NIFB_INDEX_SUBINDEX` macro, the index specified is the relative index of the parameter within the block. Relative indices start at one for the first parameter.
- In all descriptor cases, you can expand *`PARAM_NAME`* itself to *`STRUCT.ELEMENT`* format to represent a named element of a named structure.

Refer to Table 2-4 at the end of this chapter for an explanation of how to use macros to specify the parameter.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The device descriptor does not correspond to a VFD.
`E_SYMBOL_FILE_NOT_FOUND`	The NI-FBUS Communications Manager could not find the symbol file.
`E_ORDINAL_NUM_OUT_OF_RANGE`	The parameter is out of the device's range.
`E_OBJECT_ACCESS_UNSUPPORTED`	The device does not support write access to this object.
`E_MULTIPLE`	The NI-FBUS Communications Manager found more than one identical tag; the function failed.
`E_SM_NOT_OPERATIONAL`	The device is present, but cannot respond because it is at a default address.
`E_COMM_ERROR`	The NI-FBUS Communications Manager failed to communicate with the device.
`E_PARAMETER_CHECK`	The device reported a violation of parameter-specific checks.
`E_EXCEED_LIMIT`	The device reported that the value exceeds the limit.
`E_WRONG_MODE_FOR_REQUEST`	The device reported that the current function block mode does not allow you to write to the parameter.
`E_WRITE_IS_PROHIBITED`	The device reported that the `WRITE_LOCK` parameter value is set. The `WRITE_LOCK` parameter prohibits writing to the `name` parameter.
`E_DATA_NEVER_WRITABLE`	The specified object is read-only.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

Using Interface Macros

This section contains tips for using the NI-FBUS Communications Manager interface macros. These macros are defined in the header file `nifbus.h`.

Table 2-4. Core Function Macros

Descriptor Type You Have	Parameter Information You Have	Macro to Use
Block Descriptor	Name	Normal Access by Name
	Name and Subindex	`NIFB_NAME_SUBINDEX`
	Relative Index within the Block	`NIFB_INDEX`
	Relative Index and Subindex	`NIFB_INDEX_SUBINDEX`
	Device description Item ID	`NIFB_ITEM`
	Device description Item ID and Subindex	`NIFB_ITEM_SUBINDEX`
Non-Block Descriptor	Name	Normal Access Using *`BLOCKTAG.PARAM`* Format
	Name and Subindex	`NIFB_BLOCK_NAME_SUBINDEX`
	Relative Index within the Block	`NIFB_BLOCK_INDEX`
	Relative Index and Subindex	`NIFB_BLOCK_INDEX_SUBINDEX`
	Device description Item ID	`NIFB_BLOCK_ITEM`
	Device description Item ID and Subindex	`NIFB_BLOCK_ITEM_SUBINDEX`

As shown in Table 2-4, you can specify the parameter your application reads in the `name` parameter in the following ways:

- To specify an object by index, use the `NIFB_INDEX` macro in the `nifbus.h` header file.
- To specify an array or structure element by index and subindex, use the `NIFB_INDEX_SUBINDEX` macro.
- If you already have a block descriptor, you can specify an object by its item ID with the `NIFB_ITEM` macro, or you can specify a subelement by its item ID with the `NIFB_ITEM_SUBINDEX` macro.

- If you do not have a block descriptor, you have the following choices:
  - You can use the `NIFB_BLOCK_ITEM` macro to specify an item.
  - You can use the `NIFB_BLOCK_ITEM_SUBINDEX` macro to specify a subelement.
  - You can use the `NIFB_BLOCK_INDEX` macro specify an object by index.
  - You can use the `NIFB_BLOCK_INDEX_SUBINDEX` macro to specify a subindex.

You can find all these macros in the `nifbus.h` header file.

Alert and Trend Functions

The following tables list the alert and trend functions.

Table 3-1. Alert Functions

Function	Purpose
`nifAcknowledgeAlarm`	Acknowledge an alarm received
`nifWaitAlert`	Wait for an alert (an event or an alarm) from a specific device or from *any* device

Table 3-2. Trend Function

Function	Purpose
`nifWaitTrend`	Wait for a trend from a specific device or from any device

nifAcknowledgeAlarm

Purpose

Acknowledge an alarm received.

Format

```
nifError_t nifAcknowledgeAlarm(nifDesc_t ud, char *alarmName)
```

Input

`ud`	A session, link, physical device, VFD, or block descriptor for the alarm.
`alarmName`	The name of the alarm object that you want the NI-FBUS Communications Manager to acknowledge. If `ud` is a block descriptor, `alarmName` should be the parameter name, otherwise `alarmName` should be in *`BLOCKTAG.PARAMNAME`* format.

Context

Block, VFD, physical device, link, session.

Description

`nifAcknowledgeAlarm` acknowledges an alarm notification from a device. The NI-FBUS Communications Manager clears the `unacknowledged` field associated with the alarm object `alarmName`.

If `ud` is a block descriptor, the `alarmName` is the same as the `alarmOrEventName` field of the alert data you get in the `nifWaitAlert` call. If `ud` is a session, link, VFD, or physical device descriptor, then `alarmName` is in *`BLOCKTAG.PARAMNAME`* format, where `blockTag` is the same as the `blockTag` field of the alert data in the `nifWaitAlert` function.

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The device descriptor is not a valid descriptor.
`E_OBJECT_ACCESS_DENIED`	The NI-FBUS Communications Manager interface does not have the required privileges. The access group you belong to is not allowed to acknowledge the event, or the password you used is wrong.
`E_COMM_ERROR`	An error occurred when the NI-FBUS Communications Manager tried to communicate with the device.
`E_ALARM_ACKNOWLEDGED`	The alarm has already been acknowledged.

`E_MULTIPLE`	There are identical block tags.
`E_NOT_FOUND`	There is no such block in the device or VFD with the specified tag.
`E_SYMBOL_FILE_NOT_FOUND`	The NI-FBUS Communications Manager could not find the symbol file.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifWaitAlert

Purpose

Wait for an alert (an event or an alarm) from a specific device or from *any* device.

Format

```
nifError_t nifWaitAlert(nifDesc_t ud, nifAlertData_t *aldata,
            uint8 alertPriority)
```

Input

`ud`	The descriptor of the session, link, physical device, VFD, block, or link the alert comes from.
`alertPriority`	Lowest priority of the alert coming in that you want to wait on.

Output

`aldata`	The information about the specific alert.

Context

Block, VFD, physical device, link, session.

Description

`ud` represents a descriptor of a session, link, a physical device, a VFD, or a block. If `ud` is a VFD descriptor, then the NI-FBUS Communications Manager waits for an alert from any block in the Virtual Field Device. If `ud` is a block, the NI-FBUS Communications Manager waits for an alarm or event from the block `ud` refers to. If `ud` represents a link, `nifWaitAlert` completes when an event is received from any device connected to that link. If the descriptor is a session descriptor, the function waits on any event from any attached link.

`nifWaitAlert` waits indefinitely until the NI-FBUS Communications Manager receives an alert with a priority greater than or equal to the input alert priority. Your application can have a dedicated thread which does `nifWaitAlert` only.

When the NI-FBUS Communications Manager interface receives an alert, the `aldata` parameter is filled in with the information about the alert. The form of `aldata->alertData` depends on the value of `aldata->alertType`. `alData->alarmOrEventName` is the name of the alarm parameter or event parameter that caused the alert. `alData->deviceTag` and `alData->blockTag` are the tags of the device and the block of the alarm, respectively.

`nifWaitAlert` sends a confirmation to the device, informing the alerting device that the alert was received. Note that this is a separate step from alert acknowledgment, which must be carried out for alarms using `nifAcknowledgeAlarm`.

If you have multiple threads waiting to receive the same alert, the NI-FBUS Communications Manager sends a copy of the alert to all the waiting threads. Your application must ensure that only one thread acknowledges any one alarm with a call to `nifAcknowledgeAlarm`. You can abort a pending `nifWaitAlert` call by closing the descriptor on which the call was made.

The `alertType` parameter can be `ALERT_ANALOG`, `ALERT_DISCRETE`, or `ALERT_UPDATE`.

`nifAlertData_t` is defined as follows:

```
typedef struct nifAlertData_t{
   uint8            alertType;
   char             deviceTag[TAG_SIZE + 1];
   char             blockTag[TAG_SIZE + 1];
   char             alarmOrEventName [TAG_SIZE + 1];
   uint8            alertKey;
   uint8            standardType;
   uint8            mfrType;
   uint8            messageType;
   uint8            priority;
   nifTime_t        timeStamp;
   uint16           subCode;
   uint16           unitIndex;
   union {
      float         floatAlarmData;
      uint8         discreteAlarmData;
      uint16        staticRevision;
   } alertData;
} nifAlertData_t;
```

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor you gave is invalid.
`E_OBSOLETE_DESC`	The input descriptor is no longer valid. It was closed before `nifWaitAlert` completed.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

nifWaitTrend

Purpose

Wait for a trend from a specific device or from any device.

Format

```
nifError_t nifWaitTrend(nifDesc_t ud, nifTrendData_t *trend)
```

Input

`ud`	The descriptor of the session, physical device, VFD, block, or link that the trend comes from.

Output

`trend`	The information about the specific trend.

Context

Block, VFD, physical device, link, session.

Description

`ud` represents a descriptor of a session, link, physical device, VFD, or block. If `ud` is a VFD descriptor, then the NI-FBUS Communications Manager waits for a trend from any block in the Virtual Field Device. If `ud` is a block, the NI-FBUS Communications Manager waits for a trend from the block `ud` identifies. If `ud` represents a link, the call completes when a trend is received from any device connected to that link. If the descriptor is a session descriptor, `nifWaitTrend` waits on any trend from any attached link.

`nifWaitTrend` waits indefinitely until the NI-FBUS Communications Manager interface receives a trend. Your application can have a dedicated thread which does `nifWaitTrend` only.

When a trend comes in, the `trend` parameter is filled in with the information about the trend. The form of `trend->trendData` depends on the value `of trend->trendType`. There are three trend types: `TREND_FLOAT`, `TREND_DISCRETE` and `TREND_BITSTRING`. If the trend type is `TREND_FLOAT`, the `trend->trendData` is a 16-element array of floating point numbers. If the trend type is `TREND_DISCRETE`, the `trend->trendData` is a 16-element array of 1-byte integers. If the trend type is `TREND_BITSTRING`, the `trend->trendData` is a 16-element array of 2-byte bit strings, which is equivalent to a 32-element array of 1-byte integers. `deviceTag` and `blockTag` are the device and block tags of the parameter that has the trend; `paramName` is the name of the parameter.

If you have multiple threads waiting to receive the same trend, the NI-FBUS Communications Manager sends a copy of the trend to all the waiting threads. You can abort a pending `nifWaitTrend` call by closing the descriptor on which the call was made.

The `trend` type can be `TREND_FLOAT`, `TREND_DISCRETE`, or `TREND_BITSTRING`. The sample type can be `SAMPLE_INSTANT` or `SAMPLE_AVERAGE`.

`nifTrendData_t` is defined as follows:

```
typedef struct nifTrendData_t {
   uint8 trendType;
   char deviceTag[TAG_SIZE + 1];
   char blockTag[TAG_SIZE + 1];
   char paramName[TAG_SIZE + 1];
   uint8 sampleType;
   uint32 sampleInterval;
   nifTime_t lastUpdate;
   uint8 status[16];
   union {
      float f[16];
      uint8 d[16];
      uint8 bs[32];
   } trendData;
} nifTrendData_t;
```

Return Values

`E_OK`	The call was successful.
`E_INVALID_DESCRIPTOR`	The descriptor you gave is not valid.
`E_SERVER_CONNECTION_LOST`	The session established with the NI-FBUS Communications Manager for this descriptor has been closed or lost.

A

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 `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 `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 `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 `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

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.
Alert	An alarm or event.
Alert function	A function that receives or acknowledges an alert.
Application	Function blocks.
Argument	A value you pass in a function call. Sometimes referred to as a parameter, but this documentation uses a different meaning for parameter, which is included in this glossary.
Array	Ordered, indexed list of data elements of the same type.
Attribute	Properties of parameters.

B

Bit string	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.
Boolean	Logical relational system having two values, each the opposite of the other, such as true and false or zero and one.

Bridge	An interface in a Fieldbus network between two different protocols.
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

Character string name	*See* Tag.
Core Function	The basic functions that the NI-FBUS Communications Manager software performs, such as reading and writing block parameters.

D

DD	*See* Device Description.
DDOD	Device Description Object Dictionary. The Device Description binary file.
DDS	*See* Device Description Service.
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 Service	A set of functions that applications use to access Device Descriptions.
Device ID	An identifier for a device that the manufacturer assigns. No two devices can have the same device ID.
Device tag	A name you assign to a Fieldbus device.
DLL	*See* Dynamic Link Library.
DMA	Direct Memory Access.
Dynamic Link Library	A library of functions and subroutines that links to an application at run time.

E

Event	An occurrence on a device that causes a Fieldbus entity to send the Fieldbus event message.

F

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.
FMS	*See* Fieldbus Messaging Specification.
FOUNDATION Fieldbus specification	The communications network specification that the Fieldbus Foundation created.
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

Header file	A C-language source file containing important definitions and function prototypes.

I

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.

L

LAS	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 ID	*See* Link identifier.
Link identifier	A number that specifies a link.
Live list	The list of all devices that are properly responding to the Pass Token.

M

Mode	Type of communication.

N

NI-FBUS Communications Manager	Software shipped with National Instruments Fieldbus interfaces that lets you read and write values. It does not include configuration capabilities.

O

Object	An element of an object dictionary.
Object attribute	A part of the machine-readable description of a Fieldbus object.
Object description	Describes data that is communicated over the Fieldbus.
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.
Object value	The actual data value associated with a Fieldbus object.

Octet	A single 8-bit value.
OD	*See* Object Dictionary.

P

Parameter	One of a set of network-visible values that makes up a function block.
Physical device	A single device residing at a unique address on the Fieldbus.
Physical device tag	A user-defined name for a physical device.
Program	A set of instructions the computer can follow, usually in a binary file format, such as a `.exe` file.

R

Resource block	A special block containing parameters that describe the operation of the device and general characteristics of a device, such as manufacturer and device name. Only one resource block per device is allowed.

S

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.
Symbol file	A Fieldbus Foundation or device manufacturer-supplied file that contains the ASCII names for all the objects in a device.

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).
Timeout	A period of time after which an error condition is raised if some event has not occurred.
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

Variable list	A list of variables you can access with a single Fieldbus transaction.
VFD	*See* Virtual Field Device.
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.

Index