Internet Support
GPIB: gpib.support@natinst.com
DAQ: daq.support@natinst.com
VXI: vxi.support@natinst.com
LabVIEW: lv.support@natinst.com
LabWindows: lw.support@natinst.com
HiQ: hiq.support@natinst.com
VISA: visa.support@natinst.com
FTP Site: ftp.natinst.com
Web Address: www.natinst.com

Bulletin Board Support
BBS United States: (512) 794-5422 or (800) 327-3077
BBS United Kingdom: 01635 551422
BBS France: 1 48 65 15 59

FaxBack Support
(512) 418-1111 or (800) 329-7177

Telephone Support (U.S.)
Tel: (512) 795-8248
Fax: (512) 794-5678 or (800) 328-2203

International Offices
Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Italy 02 48301892, Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 202 2544,
Netherlands 03480 33466, Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085,
Sweden 08 730 49 70, Switzerland 056 20 51 51, Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters
6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100
Important Information

Warranty

The AT-FBUS/H1 Hardware is warranted against defects in materials and workmanship for a period of two years from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

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 manual 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.
About This Manual
   How to Use the Manual Set .......................................................................................... vii
   Organization of This Manual ...................................................................................... viii
   Conventions Used in This Manual ............................................................................... viii
   Related Documentation .............................................................................................. ix
   Customer Communication ............................................................................................ ix

Chapter 1
Introduction
   Background .................................................................................................................. 1-1
   Description .................................................................................................................. 1-1

Chapter 2
Functional Overview
   Creation of Bus Descriptors ...................................................................................... 2-1
   Use of Callback Functions ......................................................................................... 2-1
   Asynchronous Calls and User Data ........................................................................... 2-2
   Buffer Management .................................................................................................. 2-2
   Use of Events in Windows 3.1 .................................................................................. 2-2
   Function Return Codes .............................................................................................. 2-3
   SIL Data Types ........................................................................................................ 2-3

Chapter 3
SIL Function Calls
   List of SIL Function Calls ......................................................................................... 3-1
   Administrative Calls .................................................................................................. 3-1
       silOpen ................................................................................................................... 3-3
       silClose ............................................................................................................... 3-6
       silGetMessage .................................................................................................... 3-7
       silPollForIndication ......................................................................................... 3-8
       silSetTimeout ..................................................................................................... 3-10
Table of Contents

FMS Calls...................................................................................................... 3-11
   silInitiate ........................................................................................ 3-11
   silAbort ....................................................................................... 3-13
   silRead ......................................................................................... 3-14
   silReadWithType ........................................................................ 3-16
   silWrite ...................................................................................... 3-18
   silWriteWithType ........................................................................ 3-20
   silGetOD ...................................................................................... 3-22
   silDefineVariableList .............................................................. 3-24
   silDeleteVariableList ................................................................ 3-26
   silInitGenDomainDownload .................................................... 3-27
   silGenDownloadSegment ......................................................... 3-28
   silTerminateGenDownload ...................................................... 3-30
   silInfoReport ............................................................................. 3-32
   silEvent ...................................................................................... 3-34
   silAckEvent ................................................................................. 3-36
   silAlterEventMonitoring ......................................................... 3-38
   silIdentify .................................................................................. 3-40
   silStatus ..................................................................................... 3-42

System Management Calls ............................................................................ 3-44
   silSetPDTag ............................................................................... 3-44
   silSetAddress ............................................................................. 3-46
   silClearAddress .......................................................................... 3-47
   silSMIdentify .............................................................................. 3-49
   silFindTagQuery ......................................................................... 3-51
   silFindTagReply .......................................................................... 3-53

Chapter 4

Callback Functions
   Confirmation Callback Function ..................................................... 4-1
   Indication Callback Function ...................................................... 4-2
   SilResponse Function ................................................................... 4-7

Appendix A

Sample Program ...................................................................................... A-1

Appendix B

Customer Communication ........................................................................ B-1

Tables
   Meaning of Indication Callback Parameters .................................. 4-3
This manual describes the functions that comprise the Fieldbus Stack Interface Library, and is intended for application programmers. The Fieldbus Stack Interface Library is intended for use with Windows 3.1.

This manual assumes that you are already familiar with the Windows operating system.

**How to Use the Manual Set**

Use the *Getting Started with Fieldbus* manual to install and configure your AT-FBUS/H1 board, the Fieldbus Stack Interface Library, and the NI-SHELL Function Block Shell software.

Use the *Getting Started with the H1 Fieldbus Device Interface Kit* manual to install and configure your Fieldbus Round Card.

Use the *Fieldbus Stack Interface Reference Manual* to learn about writing client application programs that interface to your AT-FBUS/H1 board.

Use the *NI-SHELL Function Block Shell Reference Manual* to learn about writing Function Block server applications which interface to your AT-FBUS/H1 board or which are embedded in the Fieldbus Round card.

Use the *Fieldbus Control Dialog User Manual* to learn to use the interactive Fieldbus dialog system with your AT-FBUS/H1 board.

Use the *NI-FMON Fieldbus Monitor User Manual* to learn to use the interactive NI-FMON Fieldbus Monitor utility with your AT-FBUS/H1 board.
Organization of This Manual

This manual is organized as follows:

- Chapter 1, *Introduction*, gives the background and a description of the Stack Interface Library.
- Chapter 2, *Functional Overview*, introduces some key concepts and provides an overview of the functional components of the Stack Interface Library.
- Chapter 3, *SIL Function Calls*, describes the Stack Interface Library function calls.
- Chapter 4, *Callback Functions*, describes the callback functions of the Stack Interface Library.
- Appendix A, *Sample Program*, contains a sample program using the Stack Interface Library.
- Appendix B, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.

Conventions Used in This Manual

The following conventions are used in this manual:

**bold**

Bold text denotes menus, menu items, or dialog box buttons or options.

**italic**

Italic text denotes emphasis, a cross reference, or an introduction to a key concept.

**bold italic**

Bold italic text denotes a note, caution, or warning.

**monospace**

Text in this font denotes text or characters that are to be literally input 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, variables, filenames, and extensions, and for statements and comments taken from program code.

< >

Angle brackets enclose the name of a key on the keyboard—for example, <PageDown>.

- 

A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-Alt-Delete>. 
Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

**Related Documentation**

The following document contains information that you may find helpful as you read this manual:
- *Fieldbus Standard for Use in Industrial Control Systems Part 2*
- *Fieldbus Foundation Specification*
- *Fieldbus Foundation System Management Services*
- *Function Block Application Process, Part 1*
- *Function Block Application Process, Part 2*

**Customer Communication**

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix B, *Customer Communication*, at the end of this manual.
Introduction

This chapter gives the background and a description of the Stack Interface Library.

Background

The Stack Interface Library is an Application Programmer’s Interface (API) for the National Instruments Fieldbus Communication Protocol Stack. Its most beneficial characteristics are as follows:

- It is an easy-to-use C language interface to the protocol stack.
- Its interface is dependent only on the protocol specification, not the implementation.

Description

The Stack Interface Library, or SIL, is a method used to interface an application to the National Instruments Fieldbus Communications Stack. The SIL allows synchronous and asynchronous calls, and has callback and event methods for synchronizing with the completion of asynchronous calls.

This release of the Stack Interface Library is in the form of a Dynamic Link Library (DLL) for Microsoft Windows 3.1. To use the interface library, you should do the following:

- Include “silext.h” in any source files that call the SIL.
- Link your application programs with the import library sil.lib.
- Make sure that the file sil.dll is in the Windows execution path when the user’s application loads.
- Use the large memory model for the application.
- Use the 1-byte structure member alignment for all structures that communicate with the SIL.
Both the Import Library and DLL were created with Microsoft Visual C++ Version 1.5. A sample program is included in Appendix A, Sample Program.
This chapter introduces some key concepts and provides an overview of the functional components of the Stack Interface Library.

Several key concepts must be introduced before the function calls are described. These concepts are:

- Creation of bus descriptors
- Use of callback functions
- Asynchronous calls and “user data”
- Buffer management
- Use of events in Windows 3.1
- Function call return codes

**Creation of Bus Descriptors**

To send and receive Fieldbus messages over a certain Fieldbus, an application program must first open the bus. The open call returns a descriptor, which identifies the open bus. This descriptor is passed to all subsequent SIL calls to tell them which Fieldbus to operate on. When the application has finished using a bus, it must call the `silClose` function to close the bus descriptor.

**Use of Callback Functions**

To support asynchronous calls and indications from the Fieldbus, two callback functions may be registered with the SIL when a bus is opened. One callback function, the *confirmation callback*, is called upon completion of any asynchronous calls. The other callback function, the *indication callback*, is called when any indications are received from the network.
Both callback functions are optional and can be assigned a NULL value. If the confirmation callback is NULL, asynchronous calls cannot be used. If the indication callback is NULL, an application has to poll the SIL to retrieve indications.

**Asynchronous Calls and User Data**

All calls to the SIL which involve waiting for a response across the Fieldbus may be called synchronously, which means that the function does not return until the call has completed entirely; or they may be called asynchronously, which means that the function returns before the call has completed.

When an asynchronous call completes, the SIL calls the confirmation callback function registered when the descriptor was opened. A user data pointer, a parameter you supply to the function, is returned to the application as a parameter of the callback function. User data is an arbitrary pointer; it may point to any data you want to uniquely identify the call. If the user data value is NULL, the call is synchronous.

**Buffer Management**

All asynchronous calls which provide buffers for the SIL to fill in with data must manage those buffers. The buffers must be valid until the asynchronous call completes. Passing of buffers allocated on a function’s local stack, for example, is invalid if the function might return before the asynchronous call completes.

Some buffers are allocated by the SIL; the application must inform the SIL when these buffers can be freed. Buffers for indication callback data are allocated by the SIL. The application must call the silResponse function when it is done processing the buffers so that the SIL may free them. This requirement is explained further in the silResponse function description.

**Use of Events in Windows 3.1**

The SIL implementation under Windows 3.1 requires the application to periodically call silGetMessage to allow the application’s callbacks to be called. The callback functions are only called during a call to silGetMessage.
The calling of `silGetMessage` can be handled in one of two ways. The SIL can be configured to send the application a message (WM_SIL_MESSAGE, defined in `silext.h`) when `silGetMessage` has to be called, or the application may call `silGetMessage` periodically.

### Function Return Codes

All SIL functions return 0 (zero) on success, and a value less than zero on failure. The error codes are defined in the header file `silext.h`. The `cnfErrorType_t` structure is used to return error information for Confirmed FMS service calls. A pointer to this type of structure must be passed as a parameter to all Confirmed FMS services. The structure contains a bit mask which indicates what error fields are present, along with the values of the error fields themselves. The Confirmed System Management calls use an unsigned 8-bit integer to return error codes.

### SIL Data Types

The SIL uses basic data types defined in the include files `types.h` and `string_t.h`. In addition, SIL-defined data types, structures, and external prototypes are defined in the include file `silext.h`. In order to use the SIL, you must include “`silext.h`” in the source files. `types.h` and `string_t.h` are automatically included from within `silext.h`.
This chapter describes the Stack Interface Library function calls.

**List of SIL Function Calls**

**Administrative Calls**

- **silOpen**
  - Open a bus descriptor
- **silClose**
  - Close a bus descriptor
- **silGetMessage**
  - Allow callbacks to occur in Windows 3.1
- **silPollForIndication**
  - Check for new indications
- **silSetTimeout**
  - Set timeout for synchronous calls

**FMS Calls**

- **silInitiate**
  - Perform FMS initiate
- **silAbort**
  - Perform FMS abort
- **silRead**
  - Perform FMS read
- **silReadWithType**
  - Perform FMS read with type
- **silWrite**
  - Perform FMS write
- **silWriteWithType**
  - Perform FMS write with type
- **silGetOD**
  - Perform FMS get-od
- **silDefineVarList**
  - Perform FMS define variable list
Chapter 3  SIL Function Calls

silInitGenDomainDownload Perform FMS Initiate Generic Domain Download

silGenDownloadSegment Perform FMS Generic Download Segment

silTerminateGenDownload Perform FMS Terminate Generic Domain Download

silInfoReport Perform FMS information report

silEvent Perform FMS event notification

silAckEvent Perform FMS event acknowledgment

silAlterEventMonitoring Perform FMS alter event condition monitoring

silIdentify Perform FMS identify

silStatus Perform FMS status

**System Management Calls**

silSetPDTag Perform SM Set Physical Device Tag

silSetAddress Perform SM Set Device Address

silClearAddress Perform SM Clear Device Address

silSMIdentify Perform SM Device Identify

silFindTagQuery Perform SM Find Tag Query

silFindTagReply Perform SM Find Tag Reply
silOpen

Purpose
Open an interface to a specified Fieldbus communications stack.

Format
```c
int32 silOpen(uint8 boardNo, uint8 reserved2, uint16 indBufSz,
               indicationFunction_t ind, confirmFunction_t conf
               silDesc_t *desc, void *osDep)
```

Includes
```c
#include "silext.h"
```

Parameters
- **IN boardNo**: Index of the board in the board configuration file. Must be in the range 0-(numboards-1).
- **IN reserved2**: Reserved for future use. Must be set to zero.
- **IN indBufSz**: The size in bytes to reserve for indication processing.
- **IN ind**: The callback function, if any, to handle indications from this bus.
- **IN conf**: The callback function, if any, to handle confirmations from this bus.
- **IN osDep**: OS-dependent parameter. See description.
- **OUT desc**: The descriptor for this bus, to be used in future calls to the SIL.

Return Values
- Zero on success, less than zero on error.
silOpen Administrative Calls

Continued

Description
This call opens an interface to a communications stack associated with the specified Fieldbus board. In Windows 3.1, this is the index of the board in the win.ini file. For more information about board configuration using win.ini, see the Getting Started with Fieldbus manual.

The “reserved2” parameter is reserved for future use, and must be set to zero to ensure proper operation of this function.

If you plan to handle indications with the callback method, you must specify the buffer size (indBufSz) for the SIL to use for indications on this connection. The SIL allocates this buffer at open time and free the buffer when this descriptor is closed. A buffer size of at least 1024 bytes is recommended if indication callbacks are to be used.

This call registers an optional indication callback to handle incoming indications and an optional confirmation callback to handle returned confirmations. The format of these callbacks is described in Chapter 4, Callback Functions.

If the indication callback parameter ind is NULL, you must call silPollForIndication (see function description later in this chapter) to handle indications. If the confirmation callback parameter conf is NULL, only synchronous calls can be used with the descriptor desc.

The osDep parameter’s use is operating-system dependent. In Windows 3.1, osDep can point to a window handle to notify the window when silGetMessage should be called. The message WM_SIL_MESSAGE is sent to the specified window when indications or confirmations come in. The message contains the descriptor value in the Windows parameter lparam. In Windows 3.1, if the osDep parameter is NULL, messages are not sent to the application, and you must call silGetMessage periodically to receive callbacks for indications or confirmations.

Note: osDep must be set to point to the valid window handle; it should not contain the actual value of the handle.

This call is synchronous.
silOpen Administrative Calls

Continued

Possible Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_RESOURCES</td>
<td>Internal buffers or structures cannot be allocated.</td>
</tr>
<tr>
<td>SIL_BUS_CONFLICT</td>
<td>This bus has already been opened.</td>
</tr>
<tr>
<td>SIL_HARDWARE_FAILURE</td>
<td>The Fieldbus board is not responding to messages.</td>
</tr>
<tr>
<td>SIL_BAD_CONFIG</td>
<td>The software configuration data for the board is invalid or incomplete.</td>
</tr>
</tbody>
</table>
silClose Administrative Calls

Purpose
Close the interface specified by the descriptor.

Format
int32 silClose(silDesc_t desc)

Includes
#include "silext.h"

Parameters
IN desc The descriptor for the bus to be closed.

Return Values
Zero on success, less than zero on error.

Description
This function tells the SIL to free any resources associated with the given descriptor. The specified descriptor can no longer be used, and any registered callbacks will no longer be called by the SIL.

This call is synchronous.

Possible Errors
SIL_BAD_DESCRIPTOR The descriptor is invalid.
silGetMessage Administrative Calls

Purpose
Function specific to Windows 3.1 for processing messages from the bus.

Format

```c
int32 silGetMessage(silDesc_t desc)
```

Includes

```
#include “silext.h”
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN desc</td>
<td>The descriptor for the bus to be checked for messages.</td>
</tr>
</tbody>
</table>

Return Values

Zero on success, less than zero on error.

Description

The meaning of this function depends on the operating system. In Windows 3.1, this function must be called to allow indication and confirmation callbacks. If this function is not called, only synchronous calls may be used; callbacks would never occur. In Windows 3.1, this function should either be called periodically, or whenever the SIL sends a message to the window specified in `silOpen`.

This function is synchronous. Callbacks to your code may occur during the call to this function.

Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>The descriptor is invalid.</td>
</tr>
<tr>
<td>SIL_BAD INDICATION</td>
<td>An error occurred processing an indication.</td>
</tr>
</tbody>
</table>
**Purpose**

Check to see if any indications have arrived on the specified bus.

**Format**

```c
int32 silPollForIndication (silDesc_t desc, uint16 *vcr, uint16 *userData, silFunctionCode_t *fcode, uint8 *needResp, uint16 *index, uint16 *subindex, uint32 *extra, void *data, uint8 *dataLen)
```

**Includes**

```c
#include "silext.h"
```

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN desc</td>
<td>The descriptor for the bus to be checked.</td>
</tr>
<tr>
<td>IN dataLen</td>
<td>Service-specific value.</td>
</tr>
<tr>
<td>OUT vcr</td>
<td>The VCR for which the indication arrived.</td>
</tr>
<tr>
<td>OUT userData</td>
<td>The invoke ID of the indication.</td>
</tr>
<tr>
<td>OUT fCode</td>
<td>The function code of the indication.</td>
</tr>
<tr>
<td>OUT needResp</td>
<td>Indicates whether or not silResponse must be called.</td>
</tr>
<tr>
<td>OUT index</td>
<td>Service-specific value.</td>
</tr>
<tr>
<td>OUT subindex</td>
<td>Service-specific value.</td>
</tr>
<tr>
<td>OUT extra</td>
<td>Service-specific value.</td>
</tr>
<tr>
<td>OUT data</td>
<td>Service-specific value.</td>
</tr>
<tr>
<td>OUT dataLen</td>
<td>Service-specific value.</td>
</tr>
</tbody>
</table>

**Return Values**

Zero on success, less than zero on error.
silPollForIndication

Continued

Description

This function polls the SIL to determine if a new indication has come in since the last call of this function. silPollForIndication is only meaningful if the indication callback function passed to silOpen was NULL. The return value is zero if an indication has arrived. The return value is less than zero if an error occurred or if no indications are available.

The application must have already allocated the buffer data, and *dataLength must be set to the size of data on entry. The SIL sets *dataLength to the actual size of any data in the indication upon return.

The service-specific parameters are described in Table 4-1, Meaning of Indication Callback Parameters.

This call is synchronous.

Possible Errors

- SIL_NO_INDICATIONS: No indications are waiting to be read.
- SIL_BAD_DESCRIPTOR: Descriptor was invalid.
- SIL_INVALID_CALL: This descriptor has an indication callback.
- SIL_BAD_INDICATION: An error occurred in decoding the next indication (that is, some sort of error in the packet occurred).
silSetTimeout Administrative Calls

Purpose
Set the synchronous call timeout for the specified descriptor.

Format
int32 silSetTimeout(silDesc_t desc, silTimeout_t tmo)

Includes
#include “silext.h”

Parameters
IN desc The descriptor for this bus.
IN tmo The timeout in milliseconds to wait for synchronous calls.

Return Values
Zero on success, less than zero on error.

Description
This function sets the timeout value that the SIL waits for a synchronous call to complete. The value specified is in milliseconds. When the timeout for the descriptor has expired, the synchronous calls return a timeout error (SIL_TIMEOUT).

A timeout does not cause an abort to be sent on the VCR when a synchronous call times out. To abort the VCR, the user must call silAbort (see function description later in this chapter).

This call is synchronous.

Possible Errors
SIL_BAD_DESCRIPTOR The descriptor is invalid.
sillInitiate

FMS Calls

Purpose

Perform an FMS initiate.

Format

```c
int32 sillInitiate (sillDesc_t desc, uint16 vcr, userData_t userData, int16 odVersion, uint16 profile, uint8 accProt, uint8 password, uint8 accGroups, bool_t *success, sillInitiateResponse_t *resp)
```

Includes

```c
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN</td>
<td>vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN</td>
<td>userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>IN</td>
<td>odVersion</td>
<td>The FMS “OD Version” Initiate service parameter.</td>
</tr>
<tr>
<td>IN</td>
<td>profile</td>
<td>The FMS “Profile” Initiate service parameter.</td>
</tr>
<tr>
<td>IN</td>
<td>accProt</td>
<td>The FMS “Access Protection” Initiate service parameter.</td>
</tr>
<tr>
<td>IN</td>
<td>password</td>
<td>The FMS “Password” Initiate service parameter.</td>
</tr>
<tr>
<td>IN</td>
<td>accGroups</td>
<td>The FMS “Access Groups” Initiate service parameter.</td>
</tr>
<tr>
<td>OUT</td>
<td>success</td>
<td>Success code of the FMS service (nonzero for success, zero for failure).</td>
</tr>
<tr>
<td>OUT</td>
<td>resp</td>
<td>The response from the other side, whether positive or negative.</td>
</tr>
</tbody>
</table>
**Return Values**

Zero on success, less than zero on error.

**Description**

This call performs the FMS initiate function to open a communications channel to an FMS entity on a remote device located across the specified descriptor. The response packet is returned in \( \text{resp} \). The value of the \( \text{resp} \) parameter takes on either the positive or negative forms depending on whether FMS returned a positive or negative response. To determine the error status of the call, first check the return value of the function call. If the return value is zero, the FMS initiate packet was sent out correctly; otherwise, an error occurred before the call was sent out. Next, check the value of \( \text{success} \). If FMS returned a positive response, \( \text{success} \) is TRUE; otherwise, \( \text{success} \) is set to FALSE. The caller can use the \( \text{userData} \) parameter to specify data that is returned to the confirmation callback. If \( \text{userData} \) is NULL, the call is synchronous.

**Possible Errors**

- **SIL_BAD_DESCRIPTOR**
  Descriptor was invalid.
- **SIL_RESOURCES**
  No RAM available.
- **SIL_NEGATIVE_CONFIRM**
  The call was synchronous and a negative confirmation was received.
silAbort FMS Calls

Purpose
Perform an FMS abort.

Format
```c
int32 silAbort(silDesc_t desc, uint16 vcr, int8 reason)
```

Includes
```c
#include "silext.h"
```

Parameters
- **IN desc** The descriptor for this bus.
- **IN vcr** The VCR under which to operate.
- **IN reason** The FMS abort “Reason Code” parameter.

Return Values
Zero on success, less than zero on error.

Description
This function performs an FMS abort service. The FMS parameter “Locally Generated” is set to FALSE, and “Abort Identifier” is set to USER to indicate that the user layer requested the abort. See the FMS specification for the values for `reason`.

This is an unconfirmed synchronous call.

Possible Errors
- **SIL_BAD_DESCRIPTOR** Descriptor was invalid.
- **SIL_RESOURCES** No RAM available.
silRead  FMS Calls

Purpose
Perform an FMS read.

Format
```c
int32 silRead (silDesc_t desc, uint16 vcr, userData_t userData,
              uint16 index, uint16 subindex, void *data, uint8
              *dataLen, cnfErrorType_t *errInfo)
```

Includes
```c
#include "silext.h"
```

Parameters
- **IN desc** The descriptor for this bus.
- **IN vcr** The VCR under which to operate.
- **IN userData** A pointer to any data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN index** The “Index” parameter to the FMS read service.
- **IN subindex** The “Subindex” parameter to the FMS read service. The service parameter is omitted if this parameter has the reserved value SIL_INVALID_SUBINDEX.
- **IN/OUT dataLen** The length of the data read by the FMS read service.
- **OUT data** The buffer to hold the data resulting from the FMS read service.
- **OUT errInfo** The FMS confirmed service error data if the service fails.
**silRead**  
**FMS Calls**  

**Continued**

**Return Values**

Zero on success, less than zero on error.

**Description**

This call performs an FMS read request on the specified VCR. The response data is returned in `data` when the confirmation callback for the descriptor has been called with the `userData` specified in this call. The `dataLength` variable must be set to the length of the data buffer on entry. When the call completes, `dataLength` is set to the length of the data read. If the call failed, the error code information returns in `errInfo`.

**Possible Errors**

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
- **SIL_NEGATIVE_CONFIRM**: The call was synchronous and a negative confirmation was received.
Purpose

Perform an FMS Read with Type.

Format

```c
int32 silReadWithType (silDesc_t desc, uint16 vcr,
    userData_t userData, uint16 index, uint16 subindex,
    silTypeDesc_t *type, uint8 *maxTypes, void *data,
    uint8 *dataLength, cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

IN desc The descriptor for this bus.
IN vcr The VCR under which to operate.
IN userData A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
IN index The “Index” parameter to the FMS Read service.
IN subindex The “Subindex” parameter to the FMS Read service. The service parameter is omitted if this parameter has the reserved value SIL_INVALID_SUBINDEX.

IN/OUT *maxTypes

IN The length of the pre-allocated array “type.”
OUT The actual number of elements returned in “type.”

IN/OUT dataLen

IN The length of the data resulting from the FMS Read With Type service.
OUT type The array of buffers to hold the type information resulting from the service.

OUT data The buffer to hold the data resulting from the FMS Read With Type service.
OUT errInfo The FMS confirmed service error data if the service fails.

Return Values

Zero on success, less than zero on error.
silReadWithType  FMS Calls

Continued

Description

This call performs an FMS Read With Type Request on the specified VCR. The response type information is placed in the type parameter.

The type parameter is an array of type description buffers. In most cases, the length of the array (passed in the maxTypes parameter) need only be one element, and the type parameter may point to a single buffer of type silTypeDesc_t. However, if the object being read is a variable list, then the array must be as long as the number of elements in the variable list. For example, to read a five-element variable list, you must allocate a five or more element type array, and set maxTypes to at least five to hold all of the data.

The response data is returned in data when the confirmation callback for the descriptor has been called with the userData specified in this call. The dataLength variable must be set to the length of the data buffer on entry. When the call completes, dataLength is set to the length of the data read. maxTypes is set to the number of elements returned in the type array. If the call failed, the error code information is returned in errInfo.

Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>Descriptor was invalid.</td>
</tr>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
<tr>
<td>SIL_NEGATIVE_CONFIRM</td>
<td>The call was synchronous and a negative confirmation was received.</td>
</tr>
</tbody>
</table>
silWrite FMS Calls

Purpose

Perform an FMS write.

Format

```c
int32 silWrite(silDesc_t desc, uint16 vcr, userData_t userData,
               uint16 index, uint16 subindex, void *data,
               uint8 dataLen, cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

- **IN desc**: The descriptor for this bus.
- **IN vcr**: The VCR under which to operate.
- **IN userData**: A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN index**: The “Index” parameter to the FMS write service.
- **IN subindex**: The “Subindex” parameter to the FMS write service. The service parameter is omitted if this parameter has the reserved value SIL_INVALID_SUBINDEX.
- **IN data**: The “Data” parameter to the FMS write service.
- **IN dataLen**: The length in bytes of the data for the FMS write service.
- **OUT errInfo**: The FMS confirmed service error data if the service fails.
silWrite FMS Calls

Continued

Return Values
Zero on success, less than zero on error.

Description
This call performs an FMS write request on the specified VCR. The confirmation callback for the descriptor is called with this userData when the call is completed. If an error occurred, the confirmation callback is informed and errInfo is set to the error. The errInfo parameter is not valid until the confirmation callback for this descriptor has been called with the userData for this particular call. The caller can use the userData parameter to specify data that returns to the confirmation callback.

Possible Errors
- SIL_BAD_DESCRIPTOR: Descriptor was invalid.
- SIL_RESOURCES: No RAM available.
- SIL_NEGATIVE_CONFIRM: The call was synchronous and a negative confirmation was received.
Purpose
Perform an FMS Write with Type.

Format

```c
int32 silWriteWithType(silDesc_t desc, uint16 vcr,
    userData_t userData, uint16 index, uint16 subindex,
    silTypeDesc_t *type, uint8 numTypes, void *data,
    uint8 dataLen, cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>IN index</td>
<td>The “Index” parameter to the FMS Write With Type service.</td>
</tr>
<tr>
<td>IN subindex</td>
<td>The “Subindex” parameter to the FMS Write With Type service. The service parameter is omitted if this parameter has the reserved value SIL_INVALID_SUBINDEX.</td>
</tr>
<tr>
<td>IN type</td>
<td>The “Type” parameter to the FMS Write With Type service.</td>
</tr>
<tr>
<td>IN numTypes</td>
<td>The number of list elements if this is a variable list, otherwise, set to 1.</td>
</tr>
<tr>
<td>IN data</td>
<td>The “Data” parameter to the FMS Write With Type service.</td>
</tr>
<tr>
<td>IN dataLen</td>
<td>The length in bytes of the data for the FMS Write With Type service.</td>
</tr>
<tr>
<td>OUT errInfo</td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

Return Values
Zero on success, less than zero on error.
silWriteWithType  FMS Calls

Continued

Description

This call performs an FMS Write With Type Request on the specified VCR.

The type parameter is an array of type description buffers. In most cases, the length of the array (passed in the numTypes parameter) need only be one element, and the type parameter may point to a single buffer of type silTypeDesc_t. However, if the object being written is a variable list, then the array must be as long as the number of elements in the variable list. For example, to write a five-element variable list, you must allocate a five or more element type array, and set numTypes to at least five to hold all of the data.

The confirmation callback for the descriptor is called with this userData when the call is completed. If an error occurred, the confirmation callback is informed, and errInfo is set to the error value. Note that the errInfo parameter is not valid until the confirmation callback for this descriptor has been called with the userData for this particular call. The parameter userData can be used by the caller to specify data that is returned to the confirmation callback.

Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>Descriptor was invalid.</td>
</tr>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
<tr>
<td>SIL_NEGATIVE_CONFIRM</td>
<td>The call was synchronous and a negative confirmation was received.</td>
</tr>
</tbody>
</table>
Purpose
Perform an FMS GetOD.

Format

```c
int32 silGetOD(silDesc_t desc, uint16 vcr, userData_t userData,
uint8 form, uint16 index, bool_t readMult,
uint8 *numObjs, void *data, uint8 *dataLen,
bool_t *moreFollows, cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN</td>
<td>vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN</td>
<td>userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>IN</td>
<td>form</td>
<td>The “All attributes” parameter to the FMS GetOD service.</td>
</tr>
<tr>
<td>IN</td>
<td>index</td>
<td>The “Index” parameter to the FMS GetOD service.</td>
</tr>
<tr>
<td>IN</td>
<td>readMult</td>
<td>This parameter specifies whether to read a single OD entry or multiple OD entries. A value of zero indicates a single entry, while nonzero indicates multiple entries.</td>
</tr>
<tr>
<td>OUT</td>
<td>numObjs</td>
<td>This parameter contains the number of entries returned by the GetOD service.</td>
</tr>
<tr>
<td>OUT</td>
<td>data</td>
<td>The “Data” parameter to hold the result of the FMS GetOD service.</td>
</tr>
</tbody>
</table>
silGetOD

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT dataLen</td>
<td>The length in bytes of the data for the FMS GetOD service.</td>
</tr>
<tr>
<td>OUT moreFollows</td>
<td>Boolean flag indicating whether more entries exist in the OD after the ones returned by this call.</td>
</tr>
<tr>
<td>OUT errInfo</td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

**Return Values**

Zero on success, less than zero on error.

**Description**

This call performs an FMS GetOD request. The parameter form specifies the short or long form of the OD. If readMult is nonzero, index is interpreted as the start index for reading the rest of the OD. The initial value for dataLength must be the length of the buffer data. The response data is placed in the data buffer when the confirmation callback for the descriptor is called, and the dataLength parameter is changed to the actual data length at that time. The caller can use the userData parameter to specify data that is returned to the confirmation callback. The numObjs parameter contains the number of responses in the returned data buffer.

If an error occurred, data is invalid and errInfo is set to the error which occurred. If readMult was TRUE, and if more responses are to follow, moreFollows is set to TRUE. In this case, to continue reading data, you must call silGetOD again, with the object dictionary index incremented by the number of objects already read.

**Possible Errors**

- **SIL_BAD_DESCRIPTOR** Descriptor was invalid.
- **SIL_RESOURCES** No RAM available.
- **SIL_NEGATIVE_CONFIRM** The call was synchronous and a negative confirmation was received.
silDefineVariableList  

Purpose

Perform an FMS Define Variable List.

Format

```c
int32 silDefineVarList(silDesc_t desc, uint16 vcr, 
  userData_t userData, uint16 indices[], 
  uint8 numIndices, uint16 *listIndex, 
  uint8 *accessProt, string_t extension, 
  cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IN desc</strong></td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td><strong>IN vcr</strong></td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td><strong>IN userData</strong></td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td><strong>IN indices</strong></td>
<td>The array of object indices to serve as the “List of Variables” parameter to the FMS Define Variable List service.</td>
</tr>
<tr>
<td><strong>IN numIndices</strong></td>
<td>The number of elements in the indices array.</td>
</tr>
<tr>
<td><strong>IN accessProt</strong></td>
<td>The access groups, access rights, and password protection for the variable list.</td>
</tr>
<tr>
<td><strong>IN extension</strong></td>
<td>The “extension” parameter to the FMS Define Variable List service.</td>
</tr>
<tr>
<td><strong>OUT listIndex</strong></td>
<td>The index of the list object created, returned by the service if successful.</td>
</tr>
<tr>
<td><strong>OUT errInfo</strong></td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

Return Values

Zero on success, less than zero on error.
silDefineVariableList FMS Calls

Continued

Description

This call performs an FMS Define Variable List request. If the call succeeds, listIndex is set to the index of the newly created variable list. The indices parameter specifies all of the indices that make up the variable list.

The accessProt parameter is a 32-bit-long bit string, encoded according to the FMS specification as follows:

- Bits 0-7: Password_bit8 - Password_bit1
- Bits 8-15: Access_groups8 - Access_groups1
- Bits 17-19: Dg, Wg, Rg
- Bits 21-23: D, W, R
- Bits 29-31: Da, Wa, Ra

The confirmation callback for the descriptor is called with this userData when the call is completed. If an error occurred, the confirmation callback is informed, and errInfo is set to the error value. Note that the errInfo parameter is not valid until the confirmation callback for this descriptor has been called with the userData for this particular call. The parameter userData can be used by the caller to specify data that is returned to the confirmation callback.

Possible Errors

- SIL_BAD_DESCRIPTOR Descriptor was invalid.
- SIL_RESOURCES No RAM available.
silDeleteVariableList  
FMS Calls

**Purpose**

Perform an FMS Define Variable List.

**Format**

```c
int32 silDeleteVarList(silDesc_t desc, uint16 vcr, 
                      userData_t userData, uint16 index, 
                      cnfErrorType_t *errInfo)
```

**Includes**

```c
#include "silext.h"
```

**Parameters**

<table>
<thead>
<tr>
<th>IN</th>
<th>desc</th>
<th>The descriptor for this bus.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN</td>
<td>userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>IN</td>
<td>index</td>
<td>The index of the variable list to be deleted.</td>
</tr>
<tr>
<td>OUT</td>
<td>errInfo</td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

**Return Values**

Zero on success, less than zero on error.

**Description**

This call performs an FMS Delete Variable List request. If successful, the list specified by `index` is deleted.

The confirmation callback for the descriptor is called with this `userData` when the call is completed. If an error occurred, the confirmation callback is informed, and `errInfo` is set to the error value. Note that the `errInfo` parameter is not valid until the confirmation callback for this descriptor has been called with the `userData` for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.

**Possible Errors**

- **SIL_BAD_DESCRIPTOR**  
  Descriptor was invalid.
- **SIL_RESOURCES**  
  No RAM available.
silInitGenDomainDownload  FMS Calls

Purpose

Perform an FMS Initiate Generic Domain Download.

Format

```c
int32 silInitGenDomainDownload (silDesc_t desc, uint16 vcr,
                                  userData_t userData, uint16 index,
                                  cnfErrorType_t *errInfo)
```

Includes

```c
#include “silext.h”
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>index</td>
<td>The index of the domain to be downloaded.</td>
</tr>
<tr>
<td>errInfo</td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

Return Values

Zero on success, less than zero on error.

Description

This call performs an FMS Initiate Generic Domain Download request.

The confirmation callback for the descriptor is called with this `userData` when the call is completed. If an error occurred, the confirmation callback is informed, and `errInfo` is set to the error value. Note that the `errInfo` parameter is not valid until the confirmation callback for this descriptor has been called with the `userData` for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.

Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>Descriptor was invalid.</td>
</tr>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
</tbody>
</table>

silGenDownloadSegment FMS Calls

Purpose
Perform an FMS Generic Download Segment.

Format

```c
int32 silGenDownloadSegment (silDesc_t desc, uint16 vcr,
   userData_t userData, uint16 index, void *data,
   uint8 dataLength, uint8 moreFollows,
   cnfErrorType_t *errInfo)
```

Includes

```
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>IN index</td>
<td>The index of the domain whose segment is to be downloaded.</td>
</tr>
<tr>
<td>IN data</td>
<td>The data representing the segment to be downloaded.</td>
</tr>
<tr>
<td>IN dataLength</td>
<td>The length of the data to be downloaded.</td>
</tr>
<tr>
<td>IN moreFollows</td>
<td>The “more follows” parameter to the FMS Generic Download Segment service.</td>
</tr>
<tr>
<td>OUT errInfo</td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

Return Values

Zero on success, less than zero on error.
Description

This call performs an FMS Generic Download Segment request. The specified segment of data is downloaded if the call succeeds.

The confirmation callback for the descriptor is called with this `userData` when the call is completed. If an error occurred, the confirmation callback is informed, and `errInfo` is set to the error value. Note that the `errInfo` parameter is not valid until the confirmation callback for this descriptor has been called with the `userData` for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.

Possible Errors

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
silTerminateGenDownload FMS Calls

Purpose

Perform an FMS Terminate Generic Download.

Format

```c
int32 silTerminateGenDownload(silDesc_t desc, uint16 vcr,
    userData_t userData, uint16 index, uint8 *result,
    cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

- **IN desc**
  - The descriptor for this bus.
- **IN vcr**
  - The VCR under which to operate.
- **IN userData**
  - A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN index**
  - The index of the domain whose download is to be terminated.
- **OUT result**
  - The “result” parameter returned by the FMS Terminate Generic Download service.
- **OUT errInfo**
  - The FMS confirmed service error data if the service fails.

Return Values

Zero on success, less than zero on error.

Description

This call performs an FMS Terminate Generic Download request. The result of the download is returned by the remote device in `result` if the call succeeds.

The confirmation callback for the descriptor is called with this `userData` when the call is completed. If an error occurred, the confirmation callback is informed, and `errInfo` is set to the error value. Note that the `errInfo` parameter is not valid until the confirmation callback for this descriptor has been called with the `userData` for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.
silTerminateGenDownload  FMS Calls

Continued

Possible Errors

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
**Purpose**

Perform an FMS Information Report.

**Format**

```c
int32 silInfoReport (silDesc_t desc, uint16 vcr, uint16 index,
                      uint16 subindex, void *data, uint8 dataLen)
```

**Includes**

```
#include “silext.h”
```

**Parameters**

- **IN desc**
  The descriptor for this bus.
- **IN vcr**
  The VCR under which to operate.
- **IN index**
  The “Index” parameter to the FMS Information Report service.
- **IN subindex**
  The “Subindex” parameter to the FMS Information Report service.
- **IN data**
  The “Data” parameter to the FMS Information Report service.
- **IN dataLen**
  The length in bytes of the data for the FMS Information Report service.

**Return Values**

Zero on success, less than zero on error.

**Description**

This call performs an FMS Information Report request. No response packets return, because this is an unconfirmed service.
Possible Errors

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
- **SIL_BUSY**: An internal error is detected.
silEvent FMS Calls

Purpose
Perform an FMS Event Notification.

Format

```
int32 silEvent (silDesc_t desc, uint16 vcr, uint16 index,
                uint16 number, void *data, uint8 length)
```

Includes

```
#include “silext.h”
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN index</td>
<td>The “Index” parameter to the FMS Event Notification service.</td>
</tr>
<tr>
<td>IN number</td>
<td>The “Event number” parameter to the FMS Event Notification service.</td>
</tr>
<tr>
<td>IN data</td>
<td>The “Data” parameter to the FMS Event Notification service.</td>
</tr>
<tr>
<td>IN dataLen</td>
<td>The length in bytes of the data for the FMS Event Notification service.</td>
</tr>
</tbody>
</table>

Return Values

Zero on success, less than zero on error.

Description

This call performs an FMS Event Notification request. The number parameter refers to the event number according to the FMS specification. No response packets is returned because this is an unconfirmed service.
Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>Descriptor was invalid.</td>
</tr>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
</tbody>
</table>
Purpose

Perform an FMS Acknowledge Event Notification.

Format

```c
int32 silAckEvent (silDesc_t desc, uint16 vcr, userData_t userData, uint16 index, uint16 number, cnfErrorType_t *errInfo)
```

Includes

```
#include "silext.h"
```

Parameters

- **IN desc**
  The descriptor for this bus.
- **IN vcr**
  The VCR under which to operate.
- **IN userData**
  A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN index**
  The “Index” parameter to the FMS Acknowledge Event Notification service.
- **IN number**
  The “Event number” parameter to the FMS Acknowledge Event Notification service.
- **OUT errInfo**
  The FMS confirmed service error data if the service fails.

Return Values

Zero on success, less than zero on error.

Description

This call performs an FMS Acknowledge Event Notification request. It should be used by a device after the device has received and processed an FMS Event Notification Indication.
Possible Errors

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
- **SIL_NEGATIVE_CONFIRM**: The call was synchronous and a negative confirmation was received.
Purpose

Perform an FMS Alter Event Condition Monitoring request.

Format

```c
int32 silAlterEventMonitoring (silDesc_t desc, uint16 vcr,
                            userData_t userData, uint16 index, bool_t enabled,
                            cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

- **IN desc**: The descriptor for this bus.
- **IN vcr**: The VCR under which to operate.
- **IN userData**: A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN index**: The “Index” parameter to the FMS Alter Event Condition Monitoring service.
- **IN enabled**: The “Enabled” parameter to the FMS Alter Event Condition Monitoring service.
- **OUT errInfo**: The FMS confirmed service error data if the service fails.

Return Values

Zero on success, less than zero on error.

Description

This call performs an FMS Alter Event Condition Monitoring request. It should be used to enable or disable a device for reporting the specified event.
### Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>Descriptor was invalid.</td>
</tr>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
<tr>
<td>SIL_NEGATIVE_CONFIRM</td>
<td>The call was synchronous and a negative confirmation was received.</td>
</tr>
</tbody>
</table>
silIdentify FMS Calls

Purpose

Perform an FMS Identify request.

Format

```c
int32 silIdentify (silDesc_t desc, uint16 vcr,
        userData_t userData, string_t *vendorName,
        string_t *modelName, string_t *revision,
        cnfErrorType_t *errInfo)
```

Includes

```c
#include "silext.h"
```

Parameters

- **IN desc**: The descriptor for this bus.
- **IN vcr**: The VCR under which to operate.
- **IN userData**: A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **OUT vendorName**: The vendor name returned by the Identify service.
- **OUT modelName**: The model name returned by the Identify service.
- **OUT revision**: The revision string returned by the Identify service.
- **OUT errInfo**: The FMS confirmed service error data if the service fails.

Return Values

Zero on success, less than zero on error.
**Description**

This call performs an FMS Identify Request on the specified VCR. The confirmation callback for the descriptor is called with this `userData` when the call is completed.

The parameters `vendorName`, `modelName`, and `revision` must point to usable buffers on input, and have their lengths set to the size of the buffers. When confirmation is received, the buffers are filled in and the lengths set accordingly. If the buffers are too small, as much of the strings as possible are copied into them.

**Possible Errors**

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
- **SIL_NEGATIVECONFIRM**: The call was synchronous and a negative confirmation was received.
**silStatus**  

**FMS Calls**

**Purpose**

Perform an FMS Status request.

**Format**

```c
int32 silStatus(silDesc_t desc, uint16 vcr, userData_t userData,
    uint8 *logicalStatus, uint8 *physicalStatus,
    uint8 *localDetailPresent, uint32 *localDetail,
    cnfErrorType_t *errInfo)
```

**Includes**

```
#include "silext.h"
```

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN</td>
<td>vcr</td>
<td>The VCR under which to operate.</td>
</tr>
<tr>
<td>IN</td>
<td>userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>OUT</td>
<td>logicalStatus</td>
<td>The logical status code returned by the Status service.</td>
</tr>
<tr>
<td>OUT</td>
<td>physicalStatus</td>
<td>The physical status code returned by the Status service.</td>
</tr>
<tr>
<td>OUT</td>
<td>localDetailPresent</td>
<td>A flag representing whether the local detail value is present.</td>
</tr>
<tr>
<td>OUT</td>
<td>localDetail</td>
<td>The local detail value (if any) returned by the Status service.</td>
</tr>
<tr>
<td>OUT</td>
<td>errInfo</td>
<td>The FMS confirmed service error data if the service fails.</td>
</tr>
</tbody>
</table>

**Return Values**

Zero on success, less than zero on error.
**silStatus**

**FMS Calls**

Continued

**Description**

This call performs an FMS status request on the specified VCR.

If the `localDetail` parameter is specified by the server, `localDetailPresent` is set to nonzero and `localDetail` is set to the bit string supplied by the server. Otherwise, `localDetailPresent` is zero and the value of `localDetail` should be ignored. If `localDetail` is supplied, only the lowest 24 bits are used. The user should mask off the upper 8 bits.

**Possible Errors**

- **SIL_BAD_DESCRIPTOR**
  - Description: Descriptor was invalid.
- **SIL_RESOURCES**
  - Description: No RAM available.
- **SIL_NEGATIVE_CONFIRM**
  - Description: The call was synchronous and a negative confirmation was received.
silSetPDTag  System Management Calls

Purpose
Perform an SM Set Physical Device Tag service.

Format

```c
int32 silSetPDTag(silDesc_t desc, userData_t userData,
                 string_t pdTag, dlAddr_t nodeAddress,
                 string_t deviceID, bool_t clear,
                 uint8 *errorCode)
```

Includes

```c
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>IN or OUT</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN</td>
<td>userData</td>
<td>A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.</td>
</tr>
<tr>
<td>IN</td>
<td>pdTag</td>
<td>The tag to use.</td>
</tr>
<tr>
<td>IN</td>
<td>nodeAddress</td>
<td>The address of the device.</td>
</tr>
<tr>
<td>IN</td>
<td>deviceID</td>
<td>The System Management ID of the device.</td>
</tr>
<tr>
<td>IN</td>
<td>clear</td>
<td>Indicates whether or not to clear the device’s tag.</td>
</tr>
<tr>
<td>OUT</td>
<td>errorCode</td>
<td>The SM error code if the service fails.</td>
</tr>
</tbody>
</table>

Return Values
Zero on success, less than zero on error.
**silSetPDTag**

**System Management Calls**

Continued

**Description**

This call performs an SM Set Physical Device Tag service. Both the node address and the device ID of the target device must be specified. If `clear` is nonzero, the `pdTag` parameter is ignored, and the device’s tag is cleared.

The confirmation callback for the descriptor is called with this `userData` when the call is completed. If an error occurred, the confirmation callback is informed, and `errInfo` is set to the error value. Note that the `errInfo` parameter is not valid until the confirmation callback for this descriptor has been called with the `userData` for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.

**Possible Errors**

- `SIL_BAD_DESCRIPTOR`  
  Descriptor was invalid.
- `SIL_RESOURCES`  
  No RAM available.
Purpose

Perform an SM Set Address service.

Format

```c
int32 silSetAddress(silDesc_t desc, userData_t userData,
                    string_t pdTag, dlAddr_t nodeAddress,
                    uint8 *errorCode)
```

Includes

```
#include "silext.h"
```

Parameters

- **IN desc**
  The descriptor for this bus.
- **IN userData**
  A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN pdTag**
  The tag of the device whose address is to be set.
- **IN nodeAddress**
  The new address of the device.
- **OUT errorCode**
  The SM error code if the service fails.

Return Values

Zero on success, less than zero on error.

Description

This call performs an SM Set Device Address service. The tag of the target device must be specified.

The confirmation callback for the descriptor is called when the call is completed. If an error occurs, the confirmation callback is informed, and `errorCode` is set to the error. Note that the `errorCode` parameter is not valid until the confirmation callback for this descriptor has been called for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.

Possible Errors

- **SIL_BAD_DESCRIPTOR**
  Descriptor was invalid.
- **SIL_RESOURCES**
  No RAM available.
silClearAddress System Management Calls

Purpose
Perform an SM Clear Address service.

Format

```c
int32 silClearAddress(silDesc_t desc, userData_t userData,
            string_t pdTag, dlAddr_t nodeAddress,
            string_t deviceID, uint8 *errorCode)
```

Includes

```
#include “silext.h”
```

Parameters

- **IN desc**
  The descriptor for this bus.
- **IN userData**
  A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN pdTag**
  The tag of the device whose address is to be cleared.
- **IN nodeAddress**
  The address of the device whose address is to be cleared.
- **IN deviceID**
  The device ID of the device whose address is to be cleared.
- **OUT errorCode**
  The SM error code if the service fails.

Return Values

Zero on success, less than zero on error.

Description

This call performs an SM Clear Device Address service. The tag, address, and ID of the target device must be specified.

The confirmation callback for the descriptor is called when the call is completed. If an error occurs, the confirmation callback is informed, and `errorCode` is set to the error. Note that the `errorCode` parameter is not valid until the confirmation callback for this descriptor has been called for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.
Continued

Possible Errors

<table>
<thead>
<tr>
<th>SIL_BAD_DESCRIPTOR</th>
<th>Descriptor was invalid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
</tbody>
</table>
silSMIdentify  System Management Calls

Purpose

Perform an SM Identify service.

Format

```
int32 silSMIdentify(silDesc_t desc, userData_t userData,
    dlAddr_t nodeAddress, string_t *pdTag,
    string_t *deviceID, uint8 *errorCode)
```

Includes

```
#include “silext.h”
```

Parameters

- **IN desc**
  The descriptor for this bus.
- **IN userData**
  A pointer to data that identifies the asynchronous call, or NULL to make the call synchronous.
- **IN nodeAddress**
  The address of the device to identify.
- **OUT pdTag**
  The tag of the device.
- **OUT deviceId**
  The device ID of the device.
- **OUT errorCode**
  The SM error code if the service fails.

Return Values

Zero on success, less than zero on error.

Description

This call performs an SM Identify service. The node address of the target device must be specified.

The confirmation callback for the descriptor is called when the call is completed. If an error occurs, the confirmation callback is informed, and `errorCode` is set to the error. Note that the `errorCode` parameter is not valid until the confirmation callback for this descriptor has been called for this particular call. The parameter `userData` can be used by the caller to specify data that is returned to the confirmation callback.
silSMIdentify System Management Calls

Continued

Possible Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL_BAD_DESCRIPTOR</td>
<td>Descriptor was invalid.</td>
</tr>
<tr>
<td>SIL_RESOURCES</td>
<td>No RAM available.</td>
</tr>
</tbody>
</table>
silFindTagQuery System Management Calls

Purpose
Perform a System Management Find Tag Query.

Format

```c
int32 silFindTagQuery (silDesc_t desc, uint8 queryID,
                       dlAddr_t destAddress, findTag_t type, string_t pdTag,
                       string_t vfdOrFBTag, uint32 paramID,
                       uint8 *errorCode)
```

Includes

```c
#include "silext.h"
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN desc</td>
<td>The descriptor for this bus.</td>
</tr>
<tr>
<td>IN queryID</td>
<td>The “Query-ID” parameter for the Find Tag service. This must be a unique identifier, because it is used to identify the Find Tag Reply indications resulting from this call.</td>
</tr>
<tr>
<td>IN destAddress</td>
<td>The “Dest-addr” service parameter.</td>
</tr>
<tr>
<td>IN type</td>
<td>The type of object being searched for: Physical Device (FIND_PD), Virtual Field Device (FIND_VFD), Function Block (FIND_FB), or parameter (FIND_FB_PARAM).</td>
</tr>
<tr>
<td>IN pdTag</td>
<td>The tag of the physical device to be located.</td>
</tr>
<tr>
<td>IN vfdOrFBTag</td>
<td>The tag of the Virtual Field Device or Function Block to be located.</td>
</tr>
<tr>
<td>IN paramID</td>
<td>The 32 bit parameter identifier of the parameter to be located.</td>
</tr>
<tr>
<td>OUT errorCode</td>
<td>The System Management service error code. This value is only valid if the service failed (in this case, the function will have returned SIL_NEGATIVE_CONFIRM).</td>
</tr>
</tbody>
</table>
silFindTagQuery  System Management Calls

Continued

Return Values

Zero on success, less than zero on error.

Description

This call performs a System Management Find Tag Query request. This call is special in that the confirmation that is returned does not contain the result of the query; it only contains information indicating whether or not the request was sent. When the other station responds to the query, a Find Tag Reply indication arrives at the descriptor's indication callback.

The queryID parameter is returned in the Find Tag Reply indication, when it occurs. The type parameter specifies the type of tag you are searching for: Physical Device tag, VFD tag, FB tag, or FB Parameter tag. The destAddress parameter specifies who to send the query to.

In the case of a VFD tag search, both the pdTag and vfdOrFBTag must be filled in. In the case of a FB tag search, or FB parameter tag search, only the vfdOrFBTag must be filled in; pdTag is ignored.

Note: This call does not wait for the first Find Tag Reply; instead, it returns when the request is sent out. The return value of the call indicates only whether the request was sent.

Possible Errors

SIL_BAD_DESCRIPTOR  If descriptor was invalid.
SIL_RESOURCES      If no RAM available.
SIL_NEGATIVE_CONFIRM If a negative confirmation occurred.
**Purpose**

Perform a System Management Find Tag reply.

**Format**

```c
int32 silFindTagReply (silDesc_t desc, uint8 queryID,
                       dlAddr_t destAddress, findTag_t type, vfdRef_t vfd,
                       uint16 odIndex, uint16 odVersion, uint8 *errorCode)
```

**Includes**

```c
#include "silext.h"
```

**Parameters**

- **IN desc**
  The descriptor for this bus.

- **IN queryID**
  The “Query-ID” parameter for the Find Tag Reply service. This must be the query ID of the Find Tag Query that this reply is for.

- **IN destAddress**
  The “Dest-addr” service parameter.

- **IN type**
  The type of object being searched for: Physical Device (FIND_PD), Virtual Field Device (FIND_VFD), Function Block (FIND_FB), or parameter (FIND_FB_PARAM).

- **IN vfd**
  The “VFD” service parameter.

- **IN odIndex**
  The “Index” service parameter.

- **IN odVersion**
  The “OD-Version” service parameter.

- **OUT errorCode**
  The System Management service error code. This value is only valid if the service failed, and the function returned SIL_NEGATIVE_CONFIRM.
silFindTagReply System Management Calls

Continued

Return Values

Zero on success, less than zero on error.

Description

This call performs a System Management Find Tag Reply request. This call should be made in response to a Find Tag Query indication, and the queryID, desc, destAddress, and type parameters should be copied from the indication parameters. The following table lists the parameters that must be filled in depending on the value of type:

<table>
<thead>
<tr>
<th>Value of type</th>
<th>Parameters to be Filled In</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND_PD</td>
<td>deviceID</td>
</tr>
<tr>
<td>FIND_VFD</td>
<td>vfd, pathInfo, moreVcr, odVersion</td>
</tr>
<tr>
<td>FIND_FB</td>
<td>vfd, pathInfo, moreVcr, odVersion, odIndex</td>
</tr>
<tr>
<td>FIND_FB_PARAM</td>
<td>vfd, pathInfo, moreVcr, odVersion, odIndex</td>
</tr>
</tbody>
</table>

For a description of each of the parameters, refer to the Fieldbus Foundation System Management Services specification. Note that the pathInfo parameter is a list of indices in the Network Management Information Base (NMIB) for VCRs that can be used to access the object of the Find Tag search. If the list is too long to be placed in a single packet, the SIL internally reduces the size of the list and sets the moreVcr parameter to TRUE. The call still succeeds.

This call is synchronous.

Note: The return value of the call indicates only whether the request was sent.
Possible Errors

- **SIL_BAD_DESCRIPTOR**: Descriptor was invalid.
- **SIL_RESOURCES**: No RAM available.
- **SIL_NEGATIVE_CONFIRM**: A negative confirmation occurred.
Callback Functions

This chapter describes the callback functions of the Stack Interface Library.

Confirmation Callback Function

Confirmations are responses from the stack about a previous call. When a confirmation enters the stack from the bus, the stack informs the SIL. If the original call was an asynchronous call, the SIL calls the confirmation callback routine, filling in the appropriate parameters for the call. In order for this to happen, the confirmation callback routine must have been previously registered with `silOpen`.

The confirmation callback routine must take the following parameters:

```c
void confirmCallback(silDesc_t desc, uint16 vcr,
                    userData_t userData, uint8 success)
```

The `userData` parameter is an arbitrary pointer that you pass in during the initial request call.

A nonzero `success` parameter indicates a successful call. If the call is successful, then any pointers passed in as part of the initial request are filled with data. If the call is unsuccessful (success is set to zero), then the `errInfo` pointer specified in the initial request (if any) is filled with the error information.
Indication Callback Function

Indications are messages from the stack about something other than the previous call. When an indication enters the stack from the bus, the stack informs the SIL. If you registered an indication callback to silOpen, then the SIL calls your indication callback routine, filling in the appropriate parameters for the call.

If the indication requires the data pointer (see Table 4-1), or if the indication requires a response from the user, the SIL sets the needResponse parameter to TRUE (nonzero). If needResponse is TRUE, you must call silResponse when indication processing is complete. This silResponse call serves the dual purpose of sending needed data to the requester, and informing the SIL that the data buffer can be reused.

**Note:** If you do not call silResponse when needResponse is nonzero, all of the reserved memory is eventually used up, and you will stop receiving indications.

The indication callback routine must take the following parameters:

```c
void indicationCallback(silDesc_t desc, uint16 vcr,
                        uint16 userData,
                        silFunctionCode_t fCode,
                        uint8 needResponse, uint16 index,
                        uint16 subindex, uint32 extra,
                        void *data, uint8 dataLength)
```

The meaning of the parameters listed after needResponse is summarized in Table 4-1, by function code. The data structures referred to are listed in the include file, silext.h.
<table>
<thead>
<tr>
<th>Function (fCode)</th>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMS_INITIATE</td>
<td>index</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to silInitiateInd_t</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Size of the data structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>silInitiateInd_t</td>
</tr>
<tr>
<td>FMS_ABORT</td>
<td>index</td>
<td>locally_generated (see FMS)</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>abort_id (see FMS)</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>reasonCode (see FMS)</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>abort_detail (see FMS)</td>
</tr>
<tr>
<td>FMS_REJECT</td>
<td>index</td>
<td>Detected locally: 0=false 1=true</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Reject PDU type (0-4) (see FMS spec)</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Reject Code (0-6) (see FMS spec)</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>userData for original call if confirmed service; otherwise NULL</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Unused</td>
</tr>
<tr>
<td>FMS_READ</td>
<td>index</td>
<td>Index of item to be read</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Subindex (if any) of item to be read</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to buffer for response</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Size of buffer for response</td>
</tr>
<tr>
<td>FMS_WRITE</td>
<td>index</td>
<td>Index of item to be written</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Subindex (if any) of item to be written</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Data to be written</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Length of data to be written</td>
</tr>
<tr>
<td>Function (fCode)</td>
<td>Parameter</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>FMS_GET_OD</td>
<td>index</td>
<td>Index (or start_index, depending on do_multiple below)</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Form (1=long form, 0=short form)</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>do_multiple (1=get multiple ODs, 0=get a single OD)</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to buffer for response. This buffer is of type silGetODResponse_t, and the data field is initialized to point to the space for your response. Size of silGetODResponse_t</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td></td>
</tr>
<tr>
<td>FMS_INFO_REPORT</td>
<td>index</td>
<td>Index</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Subindex if any</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>The info report data</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>The length of the info report data</td>
</tr>
<tr>
<td>FMS_EVENT_NOTIFY</td>
<td>index</td>
<td>Index of event</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unique event number</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>The event notify data</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>The length of the event notify data</td>
</tr>
<tr>
<td>FMS_ACK_EVENT</td>
<td>index</td>
<td>Index of event</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unique event number</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Unused</td>
</tr>
<tr>
<td>FMS_ALTER_EVENT_</td>
<td>index</td>
<td>Index of event</td>
</tr>
<tr>
<td>MONITORING</td>
<td>subindex</td>
<td>Enable/disable flag: 0=disable, nonzero=enable</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Unused</td>
</tr>
<tr>
<td>Function (fCode)</td>
<td>Parameter</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FMS_IDENTIFY</td>
<td>index</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to structure of type siliIdentifyResponseType for response data</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>sizeof (siliIdentifyResponse_t)</td>
</tr>
<tr>
<td>FMS_STATUS</td>
<td>index</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to structure of type siliStatusResponse_t</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>sizeof(siliStatusResponse_t)</td>
</tr>
<tr>
<td>SM_FIND_TAG_QUERY</td>
<td>index</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to structure of type siliFindTagQueryInd_t</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>sizeof(siliFindTagQueryInd_t)</td>
</tr>
<tr>
<td>SM_FIND_TAG_REPLY</td>
<td>index</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to structure of type siliFindTagReplyInd_t</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>sizeof(siliFindTagReplyInd_t)</td>
</tr>
<tr>
<td>SM_FB_START</td>
<td>index</td>
<td>OD Index of FB to start</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>extra</td>
<td>VFD pointer in which the FB resides</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>dataLength</td>
<td>Unused</td>
</tr>
</tbody>
</table>
Table 4-1. Meaning of Indication Callback Parameters (Continued)

<table>
<thead>
<tr>
<th>Function (fCode)</th>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FMS_READ_TYPE</strong></td>
<td>index</td>
<td>Index of the object to be read</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>Subindex (if any) of the item to be read</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Pointer to the buffer for the response. This buffer is of type</td>
</tr>
<tr>
<td></td>
<td>datalength</td>
<td>sizeof(silReadWithTypeResponse_t)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FMS_WRITE_TYPE</strong></td>
<td>index</td>
<td>Index of object to be written</td>
</tr>
<tr>
<td></td>
<td>supindex</td>
<td>Subindex (if any) of the object to be written</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Data to write to the object</td>
</tr>
<tr>
<td></td>
<td>datalength</td>
<td>Size of data to write to object</td>
</tr>
<tr>
<td><strong>FMS_DEF_VARLIST</strong></td>
<td>data</td>
<td>Pointer to the buffer for the response. This buffer is of type</td>
</tr>
<tr>
<td></td>
<td>datalength</td>
<td>sizeof(silDefineVarListInd_t)</td>
</tr>
<tr>
<td><strong>FMS_DEL_VARLIST</strong></td>
<td>index</td>
<td>Index of variable list to be deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FMS_GEN_INIT_DOWNLOAD</strong></td>
<td>index</td>
<td>Index of domain to be downloaded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FMS_GEN_DOWNLOAD_SEGMENT</strong></td>
<td>index</td>
<td>Index of the domain, one of whose segments is being downloaded</td>
</tr>
<tr>
<td></td>
<td>subindex</td>
<td>moreFollows—nonzero if more segments are to follow</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>Segment data</td>
</tr>
<tr>
<td></td>
<td>datalength</td>
<td>Number of bytes of the segment data</td>
</tr>
<tr>
<td><strong>FMS_GEN_TERM_DOWNLOAD</strong></td>
<td>index</td>
<td>Index of the domain whose download is being terminated. Notice that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calling silResponse is required for this indication, and that the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“success” parameter to silResponse will be sent as the “final result”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter in the response.</td>
</tr>
</tbody>
</table>
SilResponse Function

The `silResponse` function, provided by the SIL, has the following prototype:

```c
int32 silResponse(silDesc_t desc, uint16 vcr,
                 uint16 userData, uint8 success,
                 cnfErrorType_t *err, void *data,
                 uint8 dataLength)
```

If the `needResponse` parameter to the indication callback was set to a nonzero value, you must call `silResponse` after you have processed the indication. The first three parameters are used to identify the indication which is being responded to. Therefore, if the `desc`, `userData`, and `vcr` parameters to `silResponse` do not exactly match the descriptor, invoke ID, and VCR of a current indication, `silResponse` fails and returns a nonzero error code. Otherwise, zero is returned to indicate success.

In all cases, any response data from you is copied before the call returns, so you can safely free any allocated buffers you have after the `silResponse` call returns.

You must set the remaining `silResponse` parameters as follows (note that FMS Initiate response is a special case):

If call was successful:

FMS Initiate case:
- Set `success`=1
- Set `data` to point to the `silPositiveInitiateResponse_t` structure
  (dataLength is ignored)

All other cases:
- Set `success`=1
- Pass any response data in `data`
- Put the size of the data into `dataLength`
  (the value of `err` is ignored by the SIL)
If the call failed:

FMS Initiate case:
Set success=0
Set data to point to the
silNegativeInitiateResponse_t structure
Fill in only the errInfo portion of the structure—the remainder
is filled in by the stack
(dataLength is ignored)

All other cases:
Set success = 0
Put the error information into err
(data and dataLength are ignored)
Sample Program

This appendix contains a sample program using the Stack Interface Library.

/* siltest.c - a sample program using the Stack Interface Library */

/* Copyright 1995 National Instruments Corporation */

/*
   This program is a simple example program which exercises the common
FMS functions of the stack: Initiate, Read, Write, GetOD, Abort. It
will execute these functions in sequence in response to an operator
keystroke. The results from the Read and GetOD calls will be displayed
on the screen in hex form.

   The program also polls for incoming indications and briefly
displays them.

   This program can be compiled as a QuickWin application under Microsoft
Visual C++ 1.5. Using it with other compilers may require
some porting.
*/

#include <windows.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <silext.h>

/* defines */
#define SUBINDEX 255    /* subindex to use on read/writes */
#define INDICATION_BYTES 512  /* bytes to reserve for incoming
indications */

/* Made-up parameters to FMS initiate */
#define OD_VERSION 0x1234
#define PROFILE 0x56
#define ACCESS_PROTECTION 0
#define PASSWORD 0x78
#define ACCESS_GROUPS 0x9a

/* Made-up parameter to FMS Abort */
#define REASON_CODE 2

/* globals */
uint16 gVcr = 0;

/* forward declarations */
extern void hexDump(char *data, uint16 len);

main()
{
    silDesc_t desc1 = 0, desc2 = 0;
    int32 r;
    uint8 dataLength;
    uint8 numObjs;
    bool_t moreFollows;
    char data[255];
    silInitiateResponse_t initResp;
    silFunctionCode_t fcode;
    uint32 extra;
    cnfErrorType_t err;
    uint8 needResponse;
    bool_t success;
    uint16 index, maxIndex;
    uint16 startIndex;
    uint16 tmpsubindex;
    uint16 tmpindex;
    uint16 userData;
    uint16 vcr;
    char str[80];

    /* Open a descriptor. We specify how many bytes to hold */
    /* indications, and since we will only poll, we will pass NULL for */
    /* the callback functions, and NULL for the pointer to the */
    /* Window handle. */
    if (r = silOpen(0, 0, INDICATION_BYTES, NULL, NULL, &desc1, NULL))
    {
        printf("silOpen failed! Error code: %ld\n", r);
        exit(1);
    }
printf("silOpen succeeded. Descriptor returned: %d\n", desc1);

silSetTimeout(desc1, 3000); /* 3000 milliseconds = 3 seconds */

printf("Enter VCR number to use for FMS calls: ");
gets(str);
gVcr = atoi(str);

printf("Enter starting OD index: ");
gets(str);
index = startIndex = atoi(str);
printf("Enter highest OD index: ");
gets(str);
maxIndex = atoi(str);

while(1)
{
    dataLength = sizeof(data);
    /* check for incoming indications */
    if (silPollForIndication(desc1, &vcr, &userData, &fcode,
                              &needResponse, &tmpindex, &tmpsubindex,
                              &extra, data, &dataLength) == 0)
    {
        printf("Indication received:\n");
        printf("vcr = %d  userData = %d  fCode = %d\n", vcr, userData, fcode);
        printf("needResponse = %d  index = %d  subindex = %d ...
", needResponse, tmpindex, tmpsubindex);
        printf("dataLength = %d\n", dataLength);
    }
    printf("Press <ENTER> to send, <p> to poll, <i> for new index, <q>
     to quit\n");
    gets(str);
    if (str[0] == 'q')
        break;
    if (str[0] == 'p')
        continue;
    if (str[0] == 'i') {
        printf(" --- Enter new index: ");
        gets(str);
        index = atoi(str);
    }
    else
        printf("FMS object index = %d\n", index);
/* Try an initiate call */
printf("Trying to initiate VCR %d\n", gVcr);
r = silInitiate(desc1, gVcr, NULL, OD_VERSION, PROFILE,
                ACCESS_PROTECTION, PASSWORD, ACCESS_GROUPS,
                &success, &initResp);
if (r)
    printf("Initiate call returned error: %d\n", r);
else
    {
        if (success)
            {
                printf("Initialize was successful.\n");
                printf("verOD=0x%x  profile=0x%x  access=0x%x  passwd=0x%x
                        grps=0x%x\n",
                        initResp.pos.versionOD, initResp.pos.profileNum,
                        initResp.pos.accessProtection, initResp.pos.password,
                        initResp.pos.accessGroups);
            }
        else
            {
                printf("Initialize was rejected.\n");
                /* Print some of the response parameters */
                printf("err=0x%x  maxSendLow=%d  maxReceiveLow=%d
                        features=0x%x,%x,%x,%x,%x,%x\n",
                        initResp.neg.errorInfo, initResp.neg.maxFMSSendLow,
                        initResp.neg.maxFMSReceiveLow,
                        initResp.neg.features[0],
                        initResp.neg.features[1],
                        initResp.neg.features[2],
                        initResp.neg.features[3],
                        initResp.neg.features[4],
                        initResp.neg.features[5]);
            }
    }

/* Try a read call */
dataLength = sizeof(data);
printf("Trying to read index %d\n", index);
r = silRead(desc1, gVcr, NULL, index, SUBINDEX, data, &dataLength,
            &err);
if (r)
    printf("Read call failed: SIL error %d\n", r);
else
    {
        if (err.fieldsPresent)
{  
  printf("Negative response received. Class=%d code=%d\n",
         err.errorClass, err.errorCode);
}
else
{
  printf("Read call succeeded!\n");
  printf("dataLength = %d (0x%x)\n", dataLength, dataLength);
  printf("Data:\n");
  hexDump(data, dataLength);
}

/* Try a Get OD call */
dataLength = sizeof(data);
printf("Trying a GetOD on index %d\n", index);
r = silGetOD(desc1, gVcr, NULL, 0, index, 0, &numObjs, data,
            &dataLength, &moreFollows, &err);
if (r)
  printf("GetOD call failed: SIL error %d\n", r);
else {
  if (err.fieldsPresent) {
    printf("Negative response received. Class=%d code=%d\n",
            err.errorClass, err.errorCode);
  }
  else {
    printf("GetOD call succeeded!\n");
    printf("dataLength = %d (0x%x)\n", dataLength, dataLength);
    printf("Data:\n");
    hexDump(data, dataLength);
  }
}

/* Try a Write call */
dataLength = sizeof(data);
/* Create some data */
memset(data, 0x80, dataLength);
printf("Trying a Write on index %d\n", index);

r = silWrite(desc1, gVcr, NULL, index, SUBINDEX, data, dataLength,
           &err);
if (r) {
    printf("Write call failed: SIL error %d\n", r);
} else {
    if (err.fieldsPresent) {
        printf("Negative response received. Class=%d code=%d\n", err.errorClass, err.errorCode);
    } else {
        printf("Write call succeeded!\n");
        printf("Positive response received.\n");
    }
}

/* Try an abort call */
printf("Trying an Abort on VCR %d\n", gVcr);
r = silAbort(desc1, gVcr, REASON_CODE);
if (r) {
    printf("Abort call failed: %d\n", r);
} else {
    printf("Abort call succeeded.\n");
}

/* Increment the object index we're testing with */
if (++index > maxIndex)
    index = startIndex;
} /* close while(1) */

silClose(desc1);
return 0;
}

void hexDump(char *inData, uint16 len) {
    uint16 i;
    unsigned char *data = (unsigned char *)inData;

    for (i=0; i < len; i++) {
        printf("%02x ", data[i]);
        if (((i) & (i%10)))
            printf("\n");
    }
    printf("\n");
}
Appendix B
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the Technical Support Form before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422 or (800) 327-3077  
   Up to 14,400 baud, 8 data bits, 1 stop bit, no parity
United Kingdom: 01635 551422  
   Up to 9,600 baud, 8 data bits, 1 stop bit, no parity
France: 1 48 65 15 59  
   Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, ftp.natinst.com, as anonymous and use your Internet address, such as joesmith@anywhere.com, as your password. The support files and documents are located in the /support directories.
FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access FaxBack from a touch-tone telephone at the following numbers:

(512) 418-1111 or (800) 329-7177

E-Mail Support (currently U.S. only)

You can submit technical support questions to the appropriate applications engineering team through e-mail at the Internet addresses listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

- GPIB: gpib.support@natinst.com
- DAQ: daq.support@natinst.com
- VXI: vxi.support@natinst.com
- LabVIEW: lv.support@natinst.com
- LabWindows: lw.support@natinst.com
- HiQ: hq.support@natinst.com
- VISA: visa.support@natinst.com

Fax and Telephone Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

<table>
<thead>
<tr>
<th>Country</th>
<th>Telephone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>03 9 879 9422</td>
<td>03 9 879 9179</td>
</tr>
<tr>
<td>Austria</td>
<td>0662 45 79 90 0</td>
<td>0662 45 79 90 19</td>
</tr>
<tr>
<td>Belgium</td>
<td>02 757 00 20</td>
<td>02 757 03 11</td>
</tr>
<tr>
<td>Canada (Ontario)</td>
<td>519 622 9310</td>
<td>519 622 9311</td>
</tr>
<tr>
<td>Canada (Quebec)</td>
<td>514 694 8521</td>
<td>514 694 4399</td>
</tr>
<tr>
<td>Denmark</td>
<td>45 76 26 00</td>
<td>45 76 71 11</td>
</tr>
<tr>
<td>Finland</td>
<td>90 527 2321</td>
<td>90 502 2930</td>
</tr>
<tr>
<td>France</td>
<td>1 48 14 24 24</td>
<td>1 48 14 24 14</td>
</tr>
<tr>
<td>Germany</td>
<td>089 741 31 30</td>
<td>089 714 60 35</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>2645 3186</td>
<td>2686 8505</td>
</tr>
<tr>
<td>Italy</td>
<td>02 48301892</td>
<td>02 48301915</td>
</tr>
<tr>
<td>Japan</td>
<td>03 5472 2970</td>
<td>03 5472 2977</td>
</tr>
<tr>
<td>Korea</td>
<td>02 596 7456</td>
<td>02 596 7455</td>
</tr>
<tr>
<td>Mexico</td>
<td>95 800 010 0793</td>
<td>5 520 3282</td>
</tr>
<tr>
<td>Netherlands</td>
<td>0348 433466</td>
<td>0348 430673</td>
</tr>
<tr>
<td>Norway</td>
<td>32 84 84 00</td>
<td>32 84 86 00</td>
</tr>
<tr>
<td>Singapore</td>
<td>2265886</td>
<td>2265887</td>
</tr>
<tr>
<td>Spain</td>
<td>91 640 0085</td>
<td>91 640 0533</td>
</tr>
<tr>
<td>Sweden</td>
<td>08 730 49 70</td>
<td>08 730 43 70</td>
</tr>
<tr>
<td>Switzerland</td>
<td>056 200 51 51</td>
<td>056 200 51 55</td>
</tr>
<tr>
<td>Taiwan</td>
<td>02 377 1200</td>
<td>02 737 4644</td>
</tr>
<tr>
<td>U.K.</td>
<td>01635 523545</td>
<td>01635 523154</td>
</tr>
</tbody>
</table>
Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name ________________________________
Company ____________________________________________
Address ____________________________________________

Fax (___)__________________________ Phone (___)__________________________
Computer brand _______________________ Model _______________ Processor _______________
Operating system (include version number) ___________________
Clock Speed _________MHz  RAM _________MB  Display adapter _______________
Mouse _____yes _____no  Other adapters installed ___________________
Hard disk capacity ________MB  Brand ___________________
Instruments used ___________________________________________________
National Instruments hardware product model ________________  Revision ____________
Configuration ___________________________________________________
National Instruments software product ________________  Version ____________
Configuration ___________________________________________________
The problem is _______________________________________________________
List any error messages ______________________________________________
________________________________________________________
________________________________________________________
________________________________________________________
The following steps will reproduce the problem ____________________________________________
________________________________________________________
Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products
Interrupt Level of Hardware _____________________________________________________
DMA Channels of Hardware _____________________________________________________
Base I/O Address of Hardware ___________________________________________________

Other Products
Computer Make and Model _____________________________________________________
Microprocessor ______________________________________________________________
Clock Frequency ______________________________________________________________
Type of Video Board Installed _________________________________________________
Operating System _____________________________________________________________
Operating System Version _____________________________________________________
Operating System Mode _______________________________________________________
Programming Language _______________________________________________________
Programming Language Version ________________________________________________
Other Boards in System ________________________________________________________
Base I/O Address of Other Boards ______________________________________________
DMA Channels of Other Boards ________________________________________________
Interrupt Level of Other Boards _______________________________________________
Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: NI-SHELL Function Reference Manual
Edition Date: March 1996
Part Number: 321015B-01

Please comment on the completeness, clarity, and organization of the manual.

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

If you find errors in the manual, please record the page numbers and describe the errors.

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

Thank you for your help.
Name  
Title  
Company  
Address  
Phone  

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678