

SmartBits

Multiport Port/Stream/Layer Performance Analysis System

SmartLib User Guide

Programming Library Version 3.07

AUGUST 1999

Supporting these Development Environments

Microsoft Windows Version 3.1

Windows 95

Windows NT

UNIX

Borland C/C++

Microsoft Visual C/C++

GNU C/C++

Microsoft Visual Basic

Borland Delphi

Tcl

Netcom Systems, Inc.
(818) 676-2300 Phone
(818) 676-2700 FAX

Copyright © 1993-1999 Netcom Systems, Inc. All Rights Reserved. Printed August 1999.

Disclaimer

The information contained in this manual is the property of Netcom Systems, Inc. and is furnished for use by recipient only for the purpose stated in the Software License Agreement accompanying the user documentation. Except as permitted by such License Agreement, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without the prior written permission of Netcom Systems, Inc.

Information contained in the user documentation is subject to change without notice and does not represent a commitment on the part of Netcom Systems, Inc. Netcom Systems, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in the user documentation.

Trademarks

SmartBits is a trademark of Netcom Systems, Inc.

Warranty

Netcom Systems, Inc. warrants to recipient that hardware which it supplies with this user documentation ("Product") will be free from significant defects in materials and workmanship for a period of twelve (12) months from the date of delivery (the "Warranty Period"), under normal use and conditions.

Defective Product under warranty shall be, at Netcom Systems' discretion, repaired or replaced or a credit issued to recipient's account for an amount equal to the price paid for such Product provided that: (a) such Product is returned to Netcom Systems after first obtaining a return authorization number and shipping instructions, freight prepaid, to Netcom Systems' location in the United States; (b) recipient provide a written explanation of the defect claimed; and (c) the claimed defect actually exists and was not caused by neglect, accident, misuse, improper installation, improper repair, fire, flood, lightning, power surges, earthquake or alteration. Netcom Systems will ship repaired Product to recipient, freight prepaid, within ten (10) working days after receipt of defective Product. Except as otherwise stated, any claim on account of defective materials or for any other cause whatsoever will conclusively be deemed waived by recipient unless written notice thereof is given to Netcom Systems within the Warranty Period. Product will be subject to Netcom Systems' standard tolerances for variations.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, ALL IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, ARE HEREBY EXCLUDED, AND THE LIABILITY OF NETCOM, IF ANY, FOR DAMAGES RELATING TO ANY ALLEGEDLY DEFECTIVE PRODUCT SHALL BE LIMITED TO THE ACTUAL PRICE PAID BY YOU FOR SUCH PRODUCT. IN NO EVENT WILL NETCOM SYSTEMS BE LIABLE FOR COSTS OF PROCUREMENT OF SUBSTITUTE PRODUCTS OR SERVICES, LOST PROFITS, OR ANY SPECIAL, DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, ARISING IN ANY WAY OUT OF THE SALE AND/OR LICENSE OF PRODUCTS OR SERVICES TO RECIPIENT EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

Contents

Chapter 1: Introducing SmartLib	1
What is SmartLib?.....	1
About the SmartLib Library.....	1
About the SmartLib Document Set.....	2
Compatibility.....	2
What are SmartCards, SmartModules, and MiniModules?.....	3
Function Prefixes: ET, HT, HG, NS.....	3
Conventions Used in This Guide.....	3
Technical Support.....	4
Troubleshooting.....	4
Chapter 2: What's New	5
SmartLib Questionnaire.....	7
Document Revision History.....	7
Chapter 3: General Programming Information	9
System Requirements.....	9
Guidelines for All Environments.....	9
Establishing a Link from the PC to SmartBits.....	10
Avoiding Link Timeouts.....	11
How to Identify Hubs, Slots, and Ports.....	13
Port Mapping Modes.....	13
Hub Numbering with Multiple Chassis.....	17
Library Functions for Port Mapping Mode.....	19
Understanding Multi-user Access.....	20
Using NSCreateFrame to Define Frame Templates.....	21
Chapter 4: Compatibility of Original Functions with Cards and Card Families	25
Chapter 5: Understanding Streams	29
Two Modes Used to Generate Test Traffic.....	29
Support for Variable Field Definitions (VFDs).....	31
Chapter 6: Getting SmartMetrics Results	33
How to Set up Histograms.....	34
Chapter 7: Working with Tcl	37
Installing Tcl and SmartLib.....	37
Setting up SmartLib with Tcl.....	38
Test Driving the Tcl Shell with SmartBits.....	40
Running a Sample Script.....	41
Using the New 3.07 Tcl Interface.....	43
Comparative Usage Examples.....	43
Default Values with the Tcl Interfaces.....	45

Default Value Examples	45
Converting Existing Scripts for the New Tcl	47
Additional Information.....	48
Chapter 8: Programming in MS Windows™.....	51
Installation	51
Directory Contents.....	52
General Programming Notes for Windows	53
Developing with C/C++.....	53
Developing with Tcl	55
Developing with Delphi	56
Developing with Visual Basic	56
Chapter 9: Programming in UNIX.....	59
Installation	59
UNIX Directory Structure and Content	60
Developing with C/C++.....	61
Developing with Tcl	61
Chapter 10: Code Examples	63
Where to Find the Code Examples	63
Tcl Demo Scripts.....	64
Examples in the et1000_tcl Directory.....	64
Examples in the smartlib_tcl Directory	73
Cross-reference: Functions within Tcl Code Examples	74
C Demo Modules	79
Visual Basic (VB) Demo.....	82
Chapter 11: Using LibX for Simplified Library Control	85
System Requirements.....	85
Cards Supported	85
LibX Components.....	85
Example 1 – Loading LibX with loadx.tcl.....	86
Example 2 – set_default and set_link	89
Example 3 – Transmit and Count.....	91
Example 4 – set_capture / show_capture	95
Example 5 – set_mii / show_mii.....	97
Example 6 – set_l3 / show_l3.....	101
Example 7 – set_streamxx / show_stream	103
Example 8 – Other Stream Commands	106
Summary of LibX Procedures	107
Chapter 12: Function and Structure Reference	109
<i>Listed Alphabetically. See Index for a complete listing.</i>	
Appendix A: Original Functions for the ET-1000 Only	197
<i>Listed Alphabetically. See the index under "Original Functions for the ET-1000 Only."</i>	

Appendix B: Error Codes..... 223

Appendix C: Library Revision History 227

Appendix D: Obsolete Functions and Structures..... 247
Listed Alphabetically. See the index under "Obsolete Functions and Structures."

Index 259

Chapter 1: Introducing SmartLib

What is SmartLib?

SmartLib is a programming library that helps developers create custom test applications for Netcom Systems' SmartBits systems and the ET-1000.

You can use SmartLib to:

- Create automated applications that test specific functionality.
- Extend SmartBits testing capabilities when testing nonstandard equipment.
- Run tests in either a UNIX or Windows environment.
- Create a simplified user interface for use on a production line.

About the SmartLib Library

The SmartLib Programming Library provides test functionality for:

- Ethernet 10MB, 100MB, and Gigabit systems.
- Token Ring 4MB and 16MB systems.
- VG/AnyLan in Ethernet mode.
- ATM technologies including DS1, E1, 25MB, E3, DS3, OC-3c, and OC-12c with signaling control as well as traffic generation.
- Frame relay.
- Packet Over Sonet (POS).

Three Ways to Develop Test Applications

SmartLib offers three approaches to test application development:

- The **Original Functions**. These interface with the hardware and firmware of older SmartCards, as well as with many of the newer SmartCards, SmartModules, and MiniModules.
- The **Message Functions**. These provide a more standardized syntax to interface with the hardware and firmware of newer SmartCards, SmartModules, and MiniModules, including ATM, Frame Relay, Gigabit, Layer3, Ethernet/Fast Ethernet, and Multi-Layer.
- The **SmartAPI** test routines. These are predefined test modules that interface with both the Original Functions and Message Functions.

Four Library Interfaces

Your Software Developer's Kit includes four interfaces to the library, designed for use with C/C++, Tcl, Visual Basic, and Delphi.

About the SmartLib Document Set

SmartLib 3.07 documentation includes both printed manuals and manuals in electronic (PDF) format on the distribution CD.

For on-line manuals, look in:

<Your CD>: | Documents | Manuals | SmartLib

To view and print PDF files, use one of the Acrobat readers (for UNIX or Windows) located in:

<Your CD>: | Tools |

SmartLib User Guide

This *SmartLib User Guide* provides basic information about the SmartLib Programming Library, including library installation, programming notes, original library functions, and controller functions.

Related Documentation

The core SmartLib document set also includes:

<u>Title</u>	<u>Description</u>
<i>SmartLib Message Functions</i>	Information about SmartCard- and SmartModule-specific Message Functions, including step-by-step instructions on using the Message Functions to control SmartCards, SmartModules, and MiniModules.
<i>SmartAPIs</i>	Information about individual <i>SmartAPI</i> tests suites. Each of these corresponds to a Netcom Systems Microsoft Windows™ application, such as SmartSignaling and SmartApplications. You can add these manuals to your SmartLib binder as they become available.

Compatibility

As SmartLib evolves, Netcom Systems makes every effort to maintain compatibility, so that applications developed for older modules will work either without modification or with minimal modification. You may, however, need to recompile programs to make use of new features and product releases.

For compatibility of the Original Functions with cards in the different card families, see Chapter 3, *Compatibility of Original Functions with Cards and Card Families*. Refer to the *SmartLib Message Functions* manual for information on the compatibility of cards with the Message Functions.

NOTE Be sure to check the *readme.html* file included with each release. See also the *Document Revision History* (in this chapter) and the *Library Revision History* (Appendix C) for changes that might affect your programs.

What are SmartCards, SmartModules, and MiniModules?

SmartCards are custom-designed printed circuit boards (PCBs) that fit within a SmartBits chassis to generate, capture, and track network packet data. They fit into the SMB-1000, SMB-2000 and SMB-200 chassis.

SmartModules are made up of two PCBs within a specially designed tray that fits into the SMB-6000 chassis. SmartModules provide a higher port density than SmartCards.

MiniModules consist of only one PCB, but have a higher port density than the SmartCard. They are attached to a tray specially designed for the SMB-600.

The term *card* can refer to any SmartCard or any printed circuit board within a SmartModule, or MiniModule.

Function Prefixes: ET, HT, HG, NS

All SmartLib function names have one of four prefixes: ET, HT, HG, or NS. These indicate the compatible SmartBits systems or cards, as follows:

Function Prefix	Description
ET	Functions that interact with the SmartBits controller. They are not designed to work with SmartCards, SmartModules, and MiniModules.
HT	Functions that communicate with a single SmartCard or module.
HG	Functions that communicate with a group of SmartCards or modules.
NS	Functions that communicate with a SmartBits controller or that perform a general action for SmartCards or modules. For example, the NSCreateFrame command can be used to create a frame template for a number of different cards.

Conventions Used in This Guide

This guide uses the following typographical conventions:

- Syntax entries are written with C/C++ programming conventions, unless otherwise noted. SmartLib provides interfaces for programming languages other than C++.

Coding examples are shown in a Courier font against a shaded background.

- Function and structure names are shown in a Courier font.
- Directory and file names are shown in an Arial font.
- Notes, cautions, and very important user information are put within horizontal lines, as in the following example:

NOTE Notes include related information, tips, and precautions about the topic preceding them.

- The terms *packet* and *frame* are used interchangeably.
- The terms *card* and *module* are sometimes used to refer generally to SmartCards, SmartModules, and MiniModules for SmartBits systems.

Technical Support

Technical support is available Monday through Friday between 07:00 and 18:00 Pacific Standard Time.

Support for Released Products

To obtain technical support for any Netcom Systems product, please contact our Technical Support Department using any of the following methods:

Phone: +1 800.886.8842 (*available in the U.S. and Canada*)

+1 818.676.2589

Fax: +1 818.880.9154

E-mail: support@netcomsystems.com

In addition, the latest versions of application Help files, application notes, and software and firmware updates are available on our website at:

<http://www.netcomsystems.com>

NOTE Special pre-release product support is also available via email to erp@netcomsystems.com for the Early Release Program and email to beta@netcomsystems.com for beta test programs.

Netcom Systems Headquarters

Netcom Systems, Inc.
26750 Agoura Road
Calabasas, CA 91302
USA

Troubleshooting

If you have difficulty working with the SmartLib Programming Library, consider these pointers:

- Make sure your manual is up-to-date. Check the part number in lower right corner of the cover. The most current documentation is available at the Netcom Systems website at:
<http://www.netcomsystems.com>
- Create your programs one module at a time and test often. The Tcl programming language (provided with SmartLib) is particularly useful in this way, because it enables you to test a command without compiling. Instead, you can send function calls directly from the command line.

See *Technical Support* above to contact Netcom Systems Technical Support.

Chapter 2:

What's New

We are very pleased to bring you SmartLib 3.07 programming library. We think you will find this version solid, compatible, and improved in many ways. It is a direct response to customer surveys and conversations with SmartLib users from shops of all types and sizes. We hope you will see the improvements that you've been looking for as we strive to be your partner in network testing and verification.

NOTE Because SmartLib 3.06 was released only as beta, this section reflects changes since the SmartLib 3.05 release.

Here are some of the improvements you will find in SmartLib 3.07.

Code Improvements

- Default values for the Message Functions (less coding for the developer).
- A much simpler yet familiar Tcl interface (again, less coding).
- Multi-user support for the SMB-6000/600 and the SMB-2000.
- Support for recently released and the soon-to-be-released cards, including LAN-6100, POS-6500, LAN-6201, and ML-5710.
- Enhanced ATM stream indexes (much simpler to work with).
- Improvements to the NSCreateFrame suite of commands (simplifies traffic creation across card families).
- More extensive quality testing across more platforms than ever before.

Improved Documentation and Code Support

- Basic step-by-step instructions for most card families.
- New cross-reference tables matching each card with commands they support.
- Over thirty additional code samples installed with SmartLib, including extensive ATM samples and a directory for the new Tcl interface samples.
- New cross-references showing where to find commands in the code samples.
- An in-depth "How-to" guide for using SmartLib with Tcl.
- An extensive document revision history, including page numbers.
- Usage guides including Tcl syntax for all Message Functions.
- Thorough indexes.

SmartLib Questionnaire

We hope you agree that 3.07 succeeds in its goal: to make development with SmartLib faster, easier, and more powerful while still supporting existing scripts and programs. We look forward to

supporting you in your efforts to provide faster, more stable networks. We appreciate your business and your input.

Feel free to let our Technical Support specialists and Sales representatives know your ideas and suggestions about our SmartLib 3.07 release. Or, use the questionnaire provided on the next page to send in your comments.

Notes

- The 3.07 release does not include SmartAPI for SmartApplications or SmartAPI for SmartSignaling.
- This is the last library release to support 16-bit Windows programming.

SmartLib Questionnaire

Do you like the SmartLib 3.07 improvements? Are there ways we can serve you better? Do you have ideas or suggestions about Netcom Systems SmartLib Programming Library? You can talk to our Sales force, Technical Support, or take a moment to fill out this questionnaire and send it in. We are committed to being your partner in network testing.

Date:	
Your Name:	
How do use SmartLib?	
Company Name:	
Phone Number:	
E-mail Address:	
Mailing Address:	
Your O/S & hardware:	
Degree of experience with SmartLib:	
Number of SmartBits chassis:	
Typical SmartCards you work with:	
What has your experience with SmartLib 3.07 been like?	
Do you have any requests or suggestions for improvement?	

<p>More Comments:</p>	
-----------------------	--

Please send your comments to:

Netcom Systems, Inc.
c/o SmartLib Product Manager
26750 Agoura Road
Calabasas, CA 91302

Document Revision History

This revision of the *SmartLib User Guide* (publication date August 1999) incorporates the following additions and corrections.

- For **Corrections**, page numbers are shown for both the previous release (**3.05**, publication date February 1999) and the current document (**3.07**).
- For **Additions**, page numbers refer to the current document.

Corrections

Corrections	Page	
	3.05	3.07
Corrected int Encap to int iEncap in NSCreateFrame.	60	198
Corrected description of VFD3 values for the int Range field in the HTVFDStructure data structure.	64	154
Corrected spelling of iData[n] fields in HTVFDStructure data structure.	65	154
Added byte length for Gigabit and other SmartCards in int iDataCount description in HTVFDStructure.	65	154
Corrected Syntax and Return Values for ETGetController description.	85	122
Corrected CountStructure name in ETGetCounters description.	85	207
Corrected ETCOM[n] descriptions in ETLink function.	95	125
Corrected Return Value descriptions for HTSeparateHubCommands function.	167	180
Corrected formatting and references in HTVFD function.	179	196
Corrected FrameSpec to FrameSpec_Type in NSCreateFrame.	181	198
Corrected compatibility statements in descriptions of Per-Connection Burst Count and Per-Port Burst Count to include ATM-1 as well as ATM-2.	197	227
Added new card models and integer values in HTGetCardModel.	146	164
Corrected piData type to "long" (was "unsigned long") in HTGetEnhancedStatus.	148	166
Obsoleted the HTGroupStart, HTGroupStep, and HTGroupStop commands.	154-55	262
Corrected Syntax description of NSCreateFrameAndPayload.	182	189
Corrected location of Show.tcl file (it is in Tcl/TclFiles directory, not AllCards).	21	52
Corrected Description in HTGetEnhancedStatus (...information is placed into the long ...).	148	166
Updated Return values in ETGetController.	85	122

Additions

Description	Page
<i>Understanding Multi-user Access</i>	20
<i>Establishing a Link from the PC to SmartBits</i>	10
<i>Port Mapping Modes</i>	13
<i>Using NSCreateFrame to Define Frame Templates</i>	21
<i>Compatibility of Original Functions with Cards and Card Families</i>	25
<i>Moved Understanding Streams to User Guide from Message Functions manual.</i>	29
<i>Working with Tcl</i>	37

Description	Page
<i>Cross-reference to Examples and Functions and Cross-reference of Functions to Code Examples</i>	63 / 73
<i>Visual Basic (VB) Demo</i>	82
<i>Using LibX for Simplified Library Control</i>	85
Moved descriptions of data structures to reside with their related function(s).	<i>All</i>
<i>Added the following new functions:</i>	
HTSlotOwnership	180
HTSlotRelease	181
HTSlotReserve	181
NSDisableAutoDefaults	190
NSEnableAutoDefaults	190
NSGetMaxHubs	190
NSGetMaxPorts	191
NSGetMaxSlots	191
NSGetNumHubs	191
NSGetNumPorts	191
NSGetNumSlots	192
NSSetPortMappingMode	194
NSSetDefaultsFile	193
NSSocketLink	194
NSUnLink	195
Moved all ET-1000-only Original Functions to Appendix A	197
Added new Error Codes	223
Moved Function listing from the Table of Contents to the Index	—
Removed <i>Additional Examples in This Manual</i> .	—
Added Chapter 2, <i>What's New</i> .	—
Added the <i>SmartLib Questionnaire</i>	—

Chapter 3:

General Programming Information

System Requirements

This version of SmartLib has been tested with firmware Release 10.16, the most current release of SmartBits/ET-1000 firmware at the time of writing.

You can obtain the most current release of Netcom Systems' firmware from the Netcom Systems web site. Go to:

www.netcomsystems.com

—and click on the *Support* link.

Guidelines for All Environments

Here are general programming notes for working with SmartLib. For information on programming in specific environments, see the appropriate chapters in this manual—either *Programming in MS Windows™* or *Programming in UNIX*.

Including Header Files

Source code/interface modules that call SmartLib library routines must include the appropriate header file, such as ET1000.H for C/C++ or ET1000.B32 for 32-bit Visual Basic.

Linking with Library Files

Applications that call SmartLib functions must link with the appropriate SmartLib library file. Each programming environment has a facility for configuring a list of library/shared object subdirectories. The SmartLib library file must reside in one of the directories on the library subdirectory list. Some programming environments require that this library be added to the project manually.

Byte Alignment Switch in 16-bit and 32-bit Environments

If you are compiling in 16-bit environments, the compiler switch struct member byte alignment must be set to 1 byte.

If you are compiling in a 32-bit environment, set the compiler switch struct member byte alignment to 4 bytes.

Establishing a Link from the PC to SmartBits

You can use several commands to establish a connection (either serial or IP) from the PC to the SmartBits chassis and to break the link.

Command	Function	Notes
ETLink	Establishes a serial link to an SMB 1000, SMB 2000/200, or ET-1000.	See <i>Port Mapping Modes</i> (page 13) for further information.
ETSocketLink	Establishes an IP socket connection to a SmartBits chassis. Note Use a serial-port connection to define the SmartBits IP address.	
ETUnLink	Breaks the current connection set up by using ETLink, ETSocketLink, or NSSocketLink.	
NSSocketLink	Establishes an IP socket connection to a SmartBits chassis. Optionally, in multi-user-compatible chassis, reserves either all cards or none of the cards in the chassis. Note Use a serial-port connection to define the SmartBits IP address.	See <i>Understanding Multi-user Access</i> (page 20) for further information.
NSUnLink	Breaks the current connection set up by using ETLink, ETSocketLink, or NSSocketLink.	

Use NSSocketLink in Multi-user Applications ETSocketLink, by default, reserves all slots in the chassis to the issuing user. In contrast, with NSSocketLink, you can reserve all slots or none of the slots. If you reserve none of the slots using NSSocketLink, you can then use HTSlotReserve to reserve a subset of slots and leave the other slots available to other users. This creates a multi-user environment for the SmartBits chassis. See *Understanding Multi-user Access* (page 20) for further information.

NOTE The SMB 200 chassis does not support multi-user mode.

Avoiding Link Timeouts

A serial link between the PC and SmartBits never times out. An Ethernet link, in contrast, times out after 30 minutes of inactivity. If the PC does not initiate communications for 30 minutes, SmartBits will close the socket connection. This frees the SmartBits to accept other link attempts, should the initial link connection be lost.

SmartLib Response to a Broken Link or Link Time-out

Usually a link is closed through an ETUnLink or NSUnLink command. Occasionally a link becomes broken because of network failure, power loss, or chassis time-out. If a link breaks while a SmartLib script or application is executing, the next SmartLib command that is issued attempts to elicit a response from the SMB link for 30 seconds, then reports an error.

You can increase or decrease the SmartLib timeout value with the ETSetTimeout command (see page 117).

NOTE Earlier SmartLib releases attempted to get a response for five minutes (default) before assuming a broken link.

Inserting a Keep-alive Loop in Your Application

You can avoid Ethernet link timeouts by inserting a keep-alive loop in your application. The loop should issue a command to the SmartBits chassis at a regular interval of less than 30 minutes to prevent the link from timing out.

Code examples are shown below. Note the following:

- With SmartLib 3.03 and earlier, use HTGetHubLEDs instead of ETGetLinkStatus.
- With SmartLib 3.05 and later, do *not* use HTGetHubLEDs. It will not keep the link alive with an SMB 6000.

Simple C Keep-Alive Routine

This example creates an endless loop that keeps the link alive by communicating with SmartBits every 29 minutes.

```
while (ETGetLinkStatus() >= 0) {  
  /* 29 minutes * 60 seconds/minute * 1000 millisecond  
  NSDelay(29*60*1000);  
}
```

Tcl Keep-Alive Routine

The example below creates a keep-alive loop that can be called periodically from within another loop. This allows code to run, accessing the chassis only when there has been no communication with the PC for a specified time interval.

To enable you to see results, this example runs continuously and activates `proc keepalive` every 20 seconds. In an actual keep-alive program, you should activate `proc keepalive` every 1200 to 1400 seconds (29 minutes = 1740 seconds).

```
#####  
# timeout.tcl  
proc keepalive {} {  
#Access the SMB controller so it doesn't time-out.  
ETGetLinkStatus  
puts ""  
puts "*****"  
puts "* 20 seconds have passed: Access SMB *"  
puts "*****"  
puts ""  
}  
  
# Initialize a beginning time.  
set starttime [clock seconds]  
  
# Loop for 20 seconds.  
while {1 == 1} {  
  
# Get the current time.  
set nowtime [clock seconds]  
  
# Test for values - run keepalive if 20 seconds has passed.  
if { [expr $nowtime - $starttime] > 20 } {  
keepalive  
# Reset the starttime.  
set starttime [clock seconds]  
} else {  
puts "A one second pause inserted to emulate your program running"  
after 1000  
}  
}
```

How to Identify Hubs, Slots, and Ports

To identify a port in a SmartBits chassis, you specify three values:

- **Hub #** The SmartBits chassis that contains the SmartCard or module
- **Slot #** The slot where the card or module is inserted
- **Port #** The port on the card or module

These are referred to collectively as the Hub/Slot/Port triple. For all three values, numbering starts at 0. For example, the values **iHub 0**, **iSlot 2**, and **iPort 0** specify the first hub (0), third slot (2), and first port (0).

Port Mapping Modes

To identify ports, you can use either of two port mapping modes, termed Compatible and Native. Each mode was designed to support the architecture of specific SmartBits chassis. Either mode may be used, however, with any SmartBits chassis.

Compatible Port Mapping

The Compatible mode can be used with any SmartBits chassis. It applied originally to the SMB 1000 and SMB 2000 chassis. In these systems, each slot holds one SmartCard with one port. Thus the hub and slot numbers were enough to identify a port.

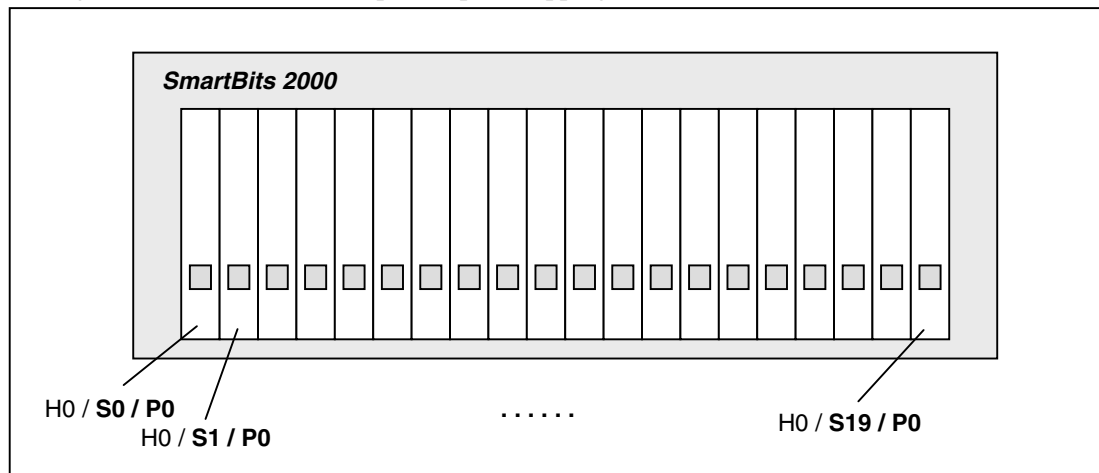
In the Compatible mode, the Hub/Slot/Port triple consists of:

- **iHub** 0, 1, 2, or 3
- **iSlot** 0 – 19
- **iPort** 0

Up to four hubs can be stacked, with Hub 0 is at the top of the stack. See *Hub Numbering with Multiple Chassis* for iHub values when chassis are interconnected.

Compatible Port Mapping in the SMB 2000/200

The figure below illustrates Compatible port mapping in a SmartBits 2000/200.



Compatible Port Mapping in SMB 6000/600

When used with an SMB 6000/600 chassis, the Compatible mode enables you to use the same hub/slot/port values as with an SMB 1000 or SMB 2000.

For compatibility, each SMB 6000/600 port is assigned a slot number, even though a SmartModule has multiple ports. After 20 ports (“slots”), the hub number increases by one (see figure).

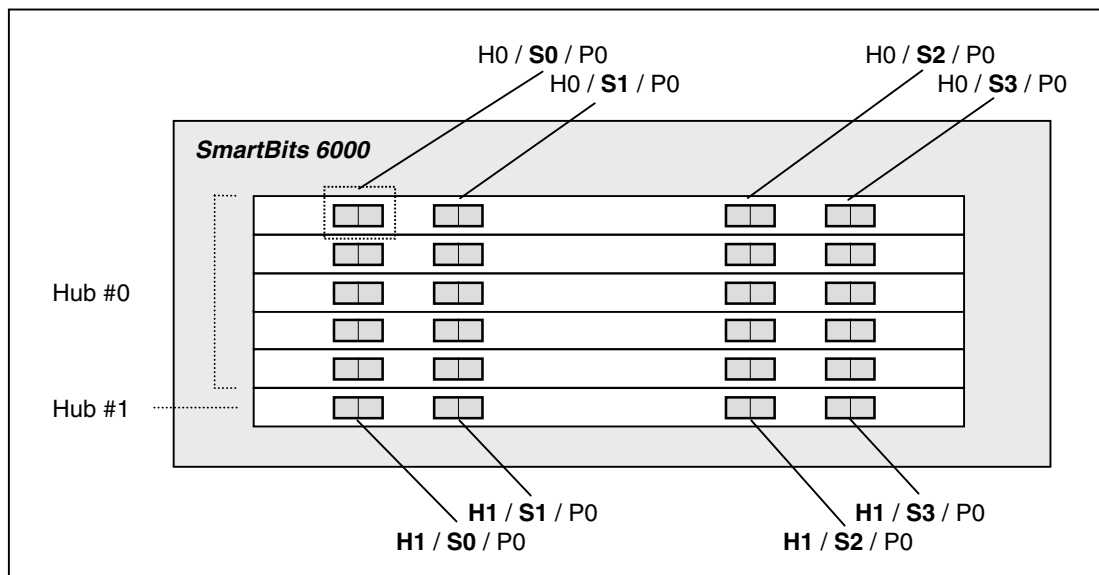
In the Compatible mode with the SMB 6000/SMB600, the Hub/Slot/Port triple consists of:

- **iHub** 0, 1, 2, or 3
- **iSlot** 0 – 19
- **iPort** 0

Numbering with Four-port SmartModules

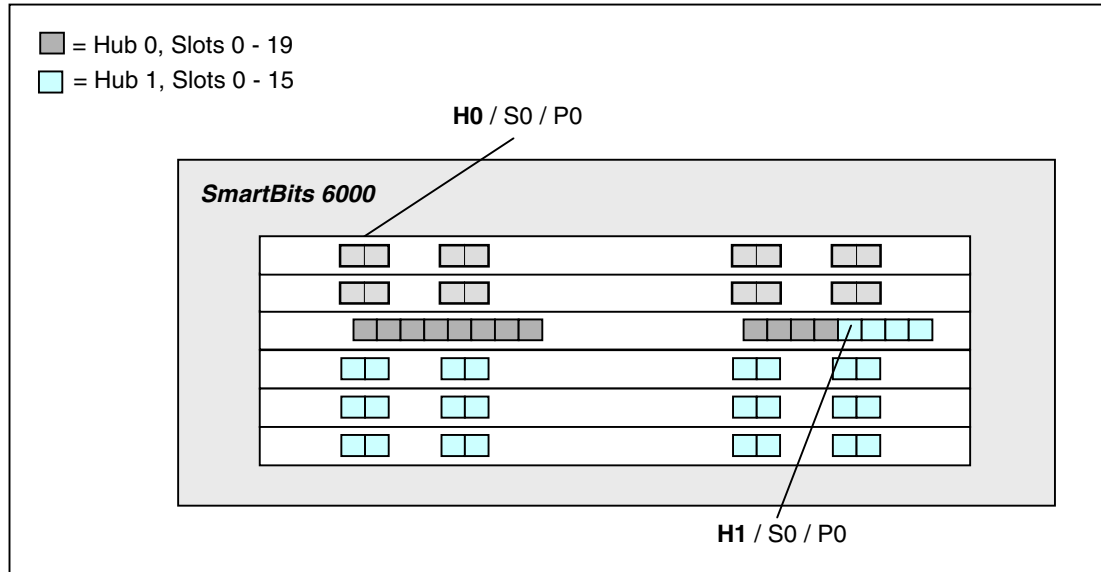
For SmartModules with four ports, Compatible port mapping in a SmartBits 6000 is as shown in the figure below. In this example:

- Slot numbering runs from top left to bottom right.
- Each SmartModule has four ports (“slots”).
- The first five SmartModules fill the first “hub” of 20 ports (H0).
- The last SmartModule begins the second hub (H1). Slot numbers start at 0 again.



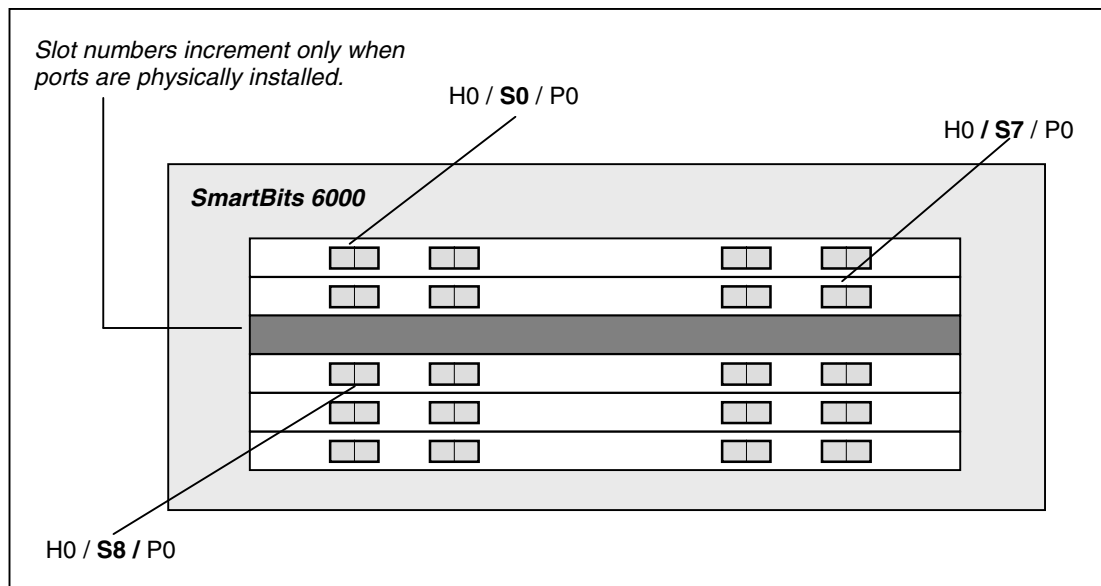
Numbering with Eight-port SmartModules

With eight-port SmartModules in an SMB6000/600, each port still counts as one slot. The hub number increments when 20 ports (“slots”) have been counted.



Numbering with Empty Slots

In Compatible mode, slot numbers increment only when ports are physically installed. If there are empty card positions and slots, the slot numbers continue with the next card or module that is physically installed.



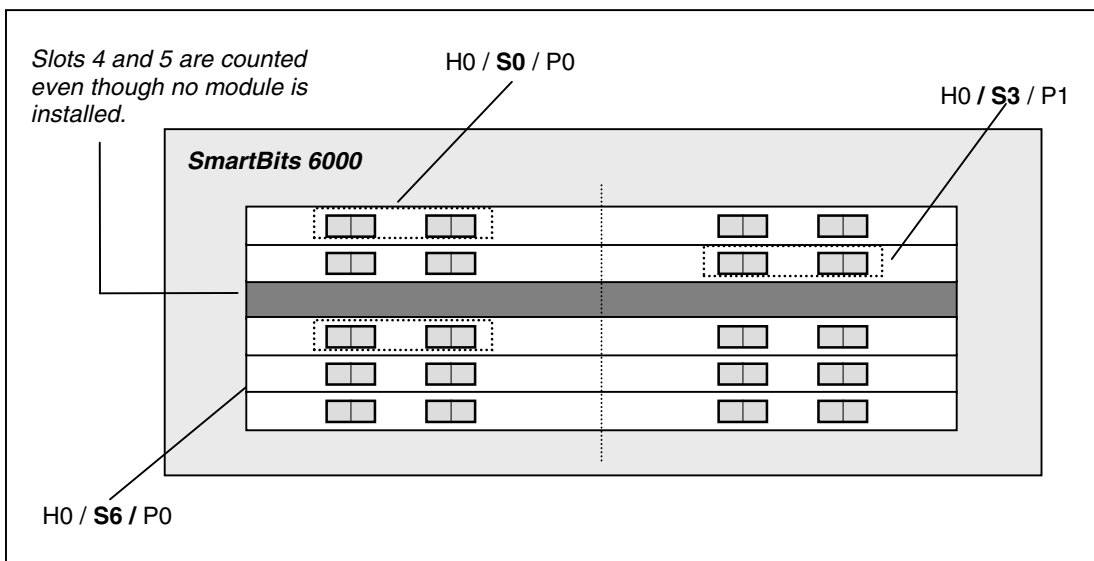
Native Port Mapping

The Native port mapping mode can be used with any SmartBits chassis. It was developed originally for the SMB 6000 chassis. In the Native mode, the Hub/Slot/Port triple consists of the following:

Value	SMB 2000/200	SMB 6000/600
iHub	0, 1, 2, 3	0
iSlot	0 through 19	0 – 1 (SMB 600) 0 – 11 (SMB 6000) The right and left sides of each SmartModule count as one slot with multiple ports.
iPort	0	0 – 7 (card-dependent)

Numbering with Empty Slots

With Native port mapping, slot numbers increment whether or not ports are physically installed. In the example below, slots 4 and 5 are counted even though a SmartModule is not installed.



Hub Numbering with Multiple Chassis

SMB 2000

You can interconnect SmartBits 2000 chassis in a stack of up to four hubs by using the 37-pin stacking connector on the back panel. The top chassis becomes the controlling chassis. You then use a connection from the controlling chassis to the PC to control all the hubs in the stack. In effect, the controllers of the other three hubs are disabled. (The SMB 200 chassis may not be stacked.)

SMB 6000/600

SmartBits 6000/600 chassis may not be stacked, but individual chassis can be connected to the same PC through separate controller links.

Synchronizing Chassis or Stacks

You can interconnect the controlling chassis in SMB 2000/200 and 6000/600 chassis, to synchronize all the chassis. One chassis, termed the *master* controller, is used to synchronize all the chassis. Each controlling chassis that uses the master controller's clock is considered a slave controller.

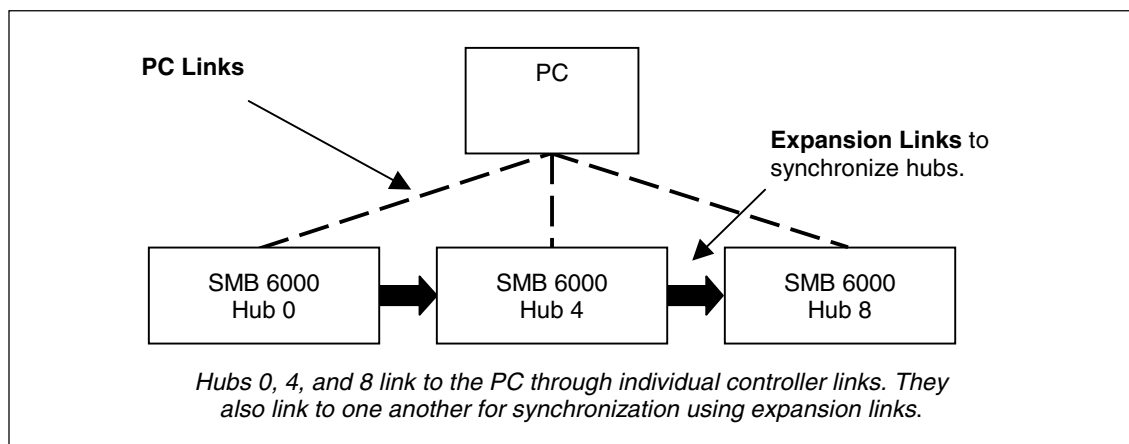
NOTE Chassis also can be synchronized through GPS connections. Refer to *Using GPS with SmartBits Chassis* for further information.

Identifying Values for Stacked Chassis

The iHub values for controller hubs start at 0 and increment by four (that is, 0, 4, 8, 12, and so on). This is true whether or not the controller hub has other hubs connected to it (using the 37-pin stacking cable), and whether or not the stack is "full" (contains all four hubs).

SMB 6000

The figure below shows four SMB 6000 chassis connected to the PC by separate controller links. Hub numbers are 0, 4, and 8.



SMB 2000

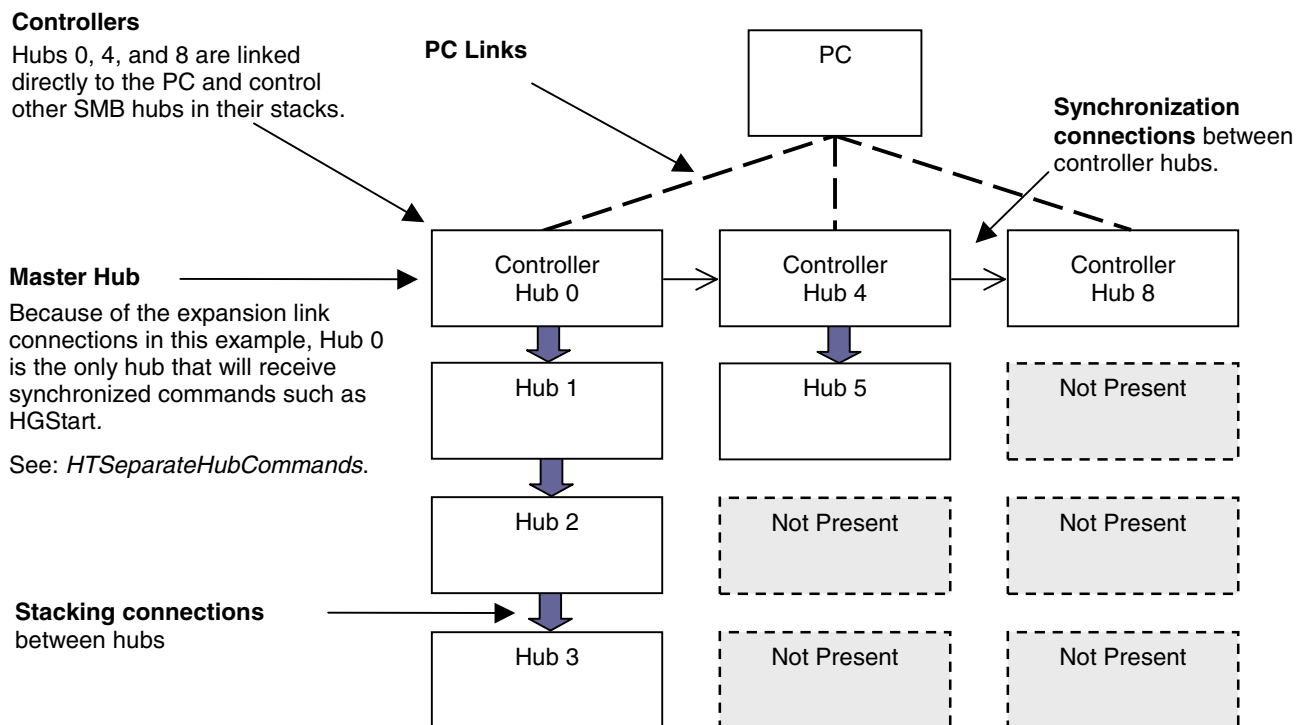
The figure below shows three stacks of SMB 2000 chassis. The chassis at the top of each stack is the controller hub for the stack. All three controller chassis are interconnected through expansion links for synchronization.

Notice that the third controller hub (third stack) is number 8 even though the previous stack contains only two chassis.

Synchronized Commands for Hub Stacks

In the figure below, only Hub 0 will receive synchronized commands such as HGStart, because the expansion connections signify that it is the master controller for all three stacks.

NOTE For the *master* controller, only the OUT expansion connector is used, not the IN expansion connector.



Library Functions for Port Mapping Mode

The following SmartLib commands are related to port mapping.

- **NSSetPortMappingMode**
This function sets the port mapping mode (either Compatible or Native) for the SmartBits chassis.
- **NSGetMaxHubs**
This function returns the maximum number of hubs per stack. It returns a value of 4 whether the connection is to a stack of SMB 2000s or an SMB 6000. This function replaces the constant MAX_SLOTS. It is useful when allocating memory.
- **NSGetMaxSlots**
This function returns the maximum number of slots per hub. It replaces the constant MAX_SLOTS. It is useful when allocating memory.
- **NSGetMaxPorts**
This function returns the maximum number of ports per slot/card. It replaces the constant MAX_PORTS. It is useful when allocating memory.
- **NSGetNumHubs**
This function returns the number of hubs possible for the chassis type. Values returned are 4 for an SMB 2000 and 1 for an SMB 6000, SMB 600, or SMB 200.
- **NSGetNumSlots**
This function returns the number of slots possible for the specified chassis (not the number of cards available).
- **NSGetNumPorts**
This function returns the number of ports possible for a specified card. SmartCards have one port per card. SmartModules have two or more ports per module.

The NSGetNum functions are useful for creating a loop to access all ports. For example:

```
for (iHub=0; iHub < NSGetNumHubs(); iHub++)
    for (iSlot=0; iSlot < NSGetNumSlots(iHub); iSlot++)
        for (iPort=0; iPort < NSGetNumPorts(iHub,iSlot); iPort++)
            HTSetStructure (iType1....    iHub,iSlot,iPort);
```

Summary of Returned Values

For the functions described above, SmartBits returns the following values for each SmartBits chassis. *Hard-coding these values into a script, rather than using the functions, reduces the portability of your script and is highly discouraged.*

Command	SMB-2000	SMB-200	SMB-6000	SMB-600
NSGetMaxHubs	4	4	4	4
NSGetMaxSlots	20	20	32	32
NSGetMaxPorts	2	2	16	16
NSGetNumHubs	4	1	1	1
NSGetNumSlots	20	4	12	2
NSGetNumPorts	Card-dependent	Card-dependent	Card-dependent	Card-dependent

Understanding Multi-user Access

SmartLib release 3.07 enables multiple users to gain access to a SmartBits 6000/600 or SmartBits 2000 chassis and reserve slots for use in different tests. (Through this discussion, “SmartBits 6000” comments apply to the SMB 600 as well.) Each user can become the owner of a subset of the cards or modules in the chassis. Reserved slots are used in tests, then may be released when no longer needed, to make them available to other users.

The multi-user capability is based on three new functions.

- **HTSlotReserve**
Reserves one or more specified slots to a user.
- **HTSlotRelease**
Releases slots that have been reserved through the HTSlotReserve function.
- **HTSlotOwnership**
Shows current slot ownership.

Two additional new, related functions are:

- **NSSocketLink**
Makes an IP connection to the SmartBits chassis while reserving either all the available slots or none of the available slots in the chassis.
- **NSUnLink**
Breaks the connection on the current link and releases all reserved slots.

How Is NSSocketLink Different from ETSocketLink?

Both NSSocketLink and ETSocketLink create an IP connection to the SmartBits chassis. ETSocketLink by default reserves all available slots in a chassis to one user. NSSocketLink, in contrast, can reserve either all available slots or none of these slots. If you use NSSocketLink and reserve none of the slots, you can use HTSlotReserve to reserve a subset of the slots (one or more), leaving the other slots available to other users.

Multi-user Access with the SMB 2000

SMB 2000 chassis with backplane revision D or later support multi-user access, using the functions described above.

NOTE If you install an SE-6205, ST-6405, or ST-6410 SmartCard in a multi-user SMB 2000, the entire chassis automatically reverts to the single-user mode.

Mixing Multi-user and Single-user Chassis

You can connect an SMB 6000/600 to an SMB 2000 with multi-user capability and use library functions in the normal way. If the SMB 2000 does not have multi-user capability, however, a connected SMB 6000 also becomes single-user (see table below). As examples, the following chassis combinations are possible:

SmartBits	Connected To	SmartBits
SMB 600 (Multi-user)	↔	SMB 6000 (Multi-user)
SMB 6000 (Multi-user)	↔	SMB 2000 (Multi-user)
SMB 6000 (Single-user)	↔	SMB 2000 (Single-user)

Multi-user and GPS

SmartBits 6000/600 and 2000/200 systems can use GPS (Global Positioning System) synchronization for end-to-end performance testing when SmartBits systems are deployed in remote locations. The multi-user capability is compatible with GPS synchronization, provided that the following guidelines are observed:

- A user requesting a GPS-synchronized action (start or stop) becomes the sole user of GPS functions during a period of about 10 seconds.
- If a second user requests a GPS-synchronized action on the same SMB chassis during the 10-second “lock-out,” SmartBits returns `MULTI_USER_CONFLICT (-37)`.
- When using GPS in a multi-user environment, set your scripts to handle such conflicts. If `MULTI_USER_CONFLICT (-37)` is encountered, create a loop to pause for 10 seconds then repeat the request for synchronized action.

Using NSCreateFrame to Define Frame Templates

`NSCreateFrame` simplifies creating test traffic for a number of different SmartCards and SmartModules. You can use `NSCreateFrame` to create test frames based on predefined frame elements, including the encapsulation type, frame size, protocol, and fill pattern. This enables you to define test frames quickly and without having to specify frame contents byte by byte. A related command, `NSModifyFrame` command, enables you to modify specified segments of the frame template.

NOTE In SmartLib Release 3.07, the `NSCreateFrame` function may not be used with frame relay cards.

Function Summary

Function	Description
<code>NSCreateFrame</code>	Create a frame template that may be used with different cards.
<code>NSCreateFrameAndPayload</code>	Performs the same function as <code>NSCreateFrame</code> also defines a custom payload (fill pattern).
<code>NSSetPayload</code>	Use with <code>NSCreateFrame</code> to define a custom payload (fill pattern).
<code>NSModifyFrame</code>	Modify specified segments of frames that were defined using <code>NSCreateFrame</code> .
<code>NSDeleteFrame</code>	Delete a frame template.
<code>HTFrame</code>	Puts specified frame elements into the card’s frame buffer.

How to Use NSCreateFrame

Use the following steps to define frame templates using the NSCreateFrame command.

Step 1 Configure the Frame (FrameSpec_Type)

Define a new data structure using FrameSpec_Type. Set values for the following:

iEncap	The frame encapsulation type.
iSize	The size of the frame template.
iProtocol	Protocol header for frame.
iPattern	Background fill pattern.

Example

```
FrameSpec_Type mySpec;  
myspec.iEncap = ENCAP_ETHERNET;  
myspec.iSize = 1560;  
myspec.iProtocol = FRAME_PROTOCOL_IP;  
myspec.iPattern = PAT_AAAA;
```

Step 2 Create the Frame Template (NSCreateFrame)

Use NSCreateFrame to create the frame template, based on the values set in the FrameSpec_Type data structure. The command returns a FrameID value that identifies the frame template.

Syntax

```
long NSCreateFrame (FrameSpec_Type* framespec);
```

Example

```
FrameID = NSCreateFrame (&mySpec);
```

Step 3 Modify Frame Elements (Optional) (NSModifyFrame)

The frame template created by the NSCreateFrame command (or NSCreateFrameAndPayload) contains the default values for the protocol type defined in your FrameSpec_Type data structure.

These values will deliver test frames to the receiving SmartCard when two SmartCards are connected in a simple back-to-back arrangement. When actually testing through a device, however, you will want to modify one or more fields in the frame to insert usable values.

For example, if your FrameSpec_Type structure specifies an IP frame, you must insert a source IP address and destination IP address to test through an attached switch or router. Use the FrameID that was returned by NSCreateFrame (or NSCreateFrameAndPayload) to identify the frame template you wish to modify.

<u>Parameters</u>	<u>Description</u>
IFrameID	Specifies which frame to modify.
iIdentifier	Specifies which element of the frame to modify (for example, destination IP address or source IP address).
pucBytes	Pointer to the replacement bytes used to modify the frame element.
iNumBytes	Length of the new segment (pucBytes). An error is returned if this value does not match the number of bytes being replaced.

Syntax

```
long NSModifyFrame (long lFrameID, int iIdentifier, unsigned char* pucBytes, int iNumBytes);
```

Example

```
char SrcIP[4] = {192,168,98,1}  
char DstIP[4] = {192,168,98,98}  
NSModifyFrame(FrameID, FRAME_DST_IP_ADDR, DstIP, 4);  
NSModifyFrame(FrameID, FRAME_SRC_IP_ADDR, SrcIP, 4);
```

Step 4 Send the Configuration the Card (HTFrame)

Use the HTFrame command to deliver the frame template to the frame buffer on the card. The FrameID identifies the frame to send.

Syntax

```
long HTFrame (long lFrameID, int iHub, int iSlot, int iPort, unsigned short uiStreamIndex);
```

Parameters

lFrameID	Identifies the frame to send.
iHub	SmartBits chassis number.
iSlot	Slot number.
iPort	Port number.
uiStreamIndex	ATM stream index. This value is used only when the protocol is ATM. With all other protocols, set it to 0 . See <i>ATM Streams and Connections</i> in the <i>SmartLib Message Functions</i> manual.

Example

Assuming that the returned FrameID is 1, the hub/slot/port values are first hub (0), second slot (1), port 1 (0).

```
HTFrame (1, 0, 1, 0, 0);
```


Chapter 4:

Compatibility of Original Functions with Cards and Card Families

This chapter outlines the compatibility of cards or card families with the Original Functions. Refer to the *SmartLib Message Functions* manual for information on compatibility among the Message Functions and card families.

Original Functions Used with All Cards

One group of Original Functions works with all cards, without exception:

- HTGetEnhancedStatus Get SmartCard status.
- HTClearPort Clear all counters on the card.
- HTRun Set SmartCard to iMode (run, step, stop).
- HGStart Start transmitting from a group of SmartCards.
- HGStop Stop transmitting from a group of SmartCards.
- HGStep Send one frame from a group of SmartCards.

Original Functions for Ethernet Cards in Traditional Mode

All Ethernet cards can use the Original Functions in the Traditional mode (single-stream definitions). This enables you to swap cards without changing test code.

Message Functions for SX-7410 Upgrade Features

All upgrade features on the SX-7410 are accessed by the Message Functions (refer to the *SmartLib Message Functions* manual). The upgrade features include:

- Alternate Transmit Stream
- Frame Capture Capability
- Software Flow Control
- Preamble Length Definition
- VLAN Tags

Exceptions for ATM SmartCards

The following two functions work on all cards *except ATM*. See the HTClearPort function (in this manual) and ATM_STREAM iType1 (in the *Message Functions* manual) for alternative ways of performing these functions.

- HTResetPort(iMode, h,s,p).
- HGResetPort(iMode).

The following tables summarize the compatibility of cards with the Original Functions. **HG** functions that are based on the equivalent **HT** function are not included. See the same-named HT function for compatibility.

Chapter 4
Compatibility of Original Functions with Cards and Card Families

Original Function	ATM	10Mbps	72xx/ 74xx	L3- 67xx	ML- 77xx	ML- 5710	LAN- 6100/ 3100	GX- 1405(B)	LAN- 6200/ 3200	LAN- 6201/ 3201	POS- 6500	Token Ring	WAN
HTAlign	N	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
HTBurstCount	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
HTBurstGap	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
HTBurstGapAndScale	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N
HTClearPort	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTCollision	N	Y	Y	N	N	N	Y	N	N	N	N	N	N
HTCollisionBackoffAggressiveness	N	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
HTCRC	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N
HTDataLength	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N
HTDuplexMode	N	6410	Y	Y	Y	Y	Y	N*	N	N	N	N	N
HTFillPattern	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTFindMIIAddress	N	Y	Y	Y	Y	Y	Y	N	N	N	N	N	Y
HTFrame	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N
HTGap	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N
HTGapAndScale	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N
HTGetCardModel	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTGetCounters	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTGetEnhancedCounters	N	Y	Y	Y	Y	Y	Y	N*	N	N	N	Y	Y
HTGetEnhancedStatus	N	6410	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y
HTGetLEDS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTGetHWVersion	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y
HTGetStructure	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y
HTHubId	Y	N	N	Y	Y	Y	N	N*	N	N	Y	Y	Y
HTHubSlotPorts	Y	Y	Y	Y	Y	Y	N	N*	N	N	Y	Y	Y
HTLayer3SetAddress	N	N	N	Y	Y	Y	N	N	N	N	Y	N	N
HTLatency	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTMultiBurstCount	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTPortProperty	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTPortType	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTReadMII	N	N	Y	Y	Y	Y	Y	N*	N	N	N	N	N
HTResetPort	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Original Function	ATM	10Mbps	72xx/ 74xx	L3- 67xx	ML- 77xx	ML- 5710	LAN- 6100/ 3100	GX- 1405(B)	LAN- 6200/ 3200	LAN- 6201/ 3201	POS- 6500	Token Ring	WAN
HTRun	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTSelectReceive	N	Y*	N	N	N	N	N	N	N	N	N	N	N
HTSelectTransmit	N	Y*	N	N	N	N	N	N	N	N	N	N	N
HTSendCommand	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTSetCommand	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HTSetSpeed	N	N*	Y	Y	Y	N	Y	N*	N	N	N	Y	N
HTSetStructure	Y	Y	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
HTSetTokenRingAdvancedControl	N	N	N	N	N	N	N	N	N	N	N	Y	N
HTSetTokenRingErrors	Y	N	N	N	N	N	N	N	N	N	N	Y	N
HTSetTokenRingLLC	N	N	N	N	N	N	N	N	N	N	N	Y	N
HTSetTokenRingMAC	N	N	N	N	N	N	N	N	N	N	N	Y	N
HTSetTokenRingProperty	N	N	N	N	N	N	N	N	N	N	N	Y	N
HTSetTokenRingSrcRouteAddr	N	N	N	N	N	N	N	N	N	N	N	Y	N
HTSetVGProperty	N	N	N	N	N	N	N	N	N	N	N	N	N
HTSymbol	N	N	Y	N	N	N	Y	0	N	N	N	N	N
HTTransmitMode	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
HTTrigger	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
HTVFD	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
HTWriteMII	N	N	Y	N	Y	Y	Y	N	N	N	N	N	N
NSCreateFrame	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N
NSCreateFrameAndPayload	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N
NSDeleteFrame	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N
NSModifyFrame	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N
NSSetPayload	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N
*ET-1000 Controller													

Chapter 5:

Understanding Streams

A stream of network traffic is a series of frames transmitted from a source address to a destination address. A stream of traffic is created in two steps: First, you define a frame blueprint. Then, using the blueprint, the SmartCard or module creates and sends multiple frames. This generates the stream of traffic.

Stream characteristics can vary with different protocols. For example, with both Ethernet and frame relay, a stream's frames may be modified so that each frame in a stream is different. With ATM, each stream combines two elements: a stream structure containing connection parameters, and a frame structure containing the payload contents.

Two Modes Used to Generate Test Traffic

Depending on the cards installed, SmartBits systems can generate test traffic using either of two modes:

- **Traditional Mode** One frame blueprint may be defined for each card.
- **SmartMetrics Mode** Multiple frame blueprints may be defined for each card.

Support for the two modes varies for different SmartCards, SmartModules, and MiniModules. Some cards support both the Traditional and SmartMetrics modes. Others support one or the other. As examples:

Traditional Mode Only. Some SmartCards support Traditional Mode exclusively. Examples are the SX-7210, SX-7410, and TR-8405.

SmartMetrics Mode Only. Some SmartCards support SmartMetrics exclusively. An example is the WN-3410.

Both Modes. Other SmartCards support both SmartMetrics and Traditional Mode. Examples are the ML-7710 and L3-6710.

Traditional Mode Traffic

In the Traditional mode, one frame blueprint is available per card. To simulate more than one stream per card, you use incrementing or varying patterns within specified fields of the frame blueprint.

The figure below is an example of an Ethernet frame blueprint in the Traditional mode. In this example, the VFD1 (Variable Field Definition 1) field—which contains the source MAC address—increments in each frame. This creates frames that seem to come from different devices.

Destination MAC Address	Source MAC Address (Incrementing VFD1)	Type	Payload (Defined Background Pattern)	CRC
-------------------------	--	------	--------------------------------------	-----

Characteristics of Traditional Mode

Some characteristics of Traditional mode are:

- Traffic is based on modifications of a single frame blueprint.
- By using the Variable Field Definitions VFD1, VFD2, and VFD3 in addition to the background (fill) pattern, you can achieve a high degree of complexity and control.
- There is a CRC check on the entire frame.
- Traffic can be used to test both Layer 2 and Layer 3 devices.
- Because there is only one frame blueprint per port, information is tracked on a per-port basis.

SmartMetrics Mode Traffic

In the SmartMetrics mode, a card can support many unique frame blueprints. Because each frame blueprint is used to generate a different stream of test traffic, the blueprints themselves are termed streams.

Characteristics of SmartMetrics Mode

Some characteristics of SmartMetrics mode are:

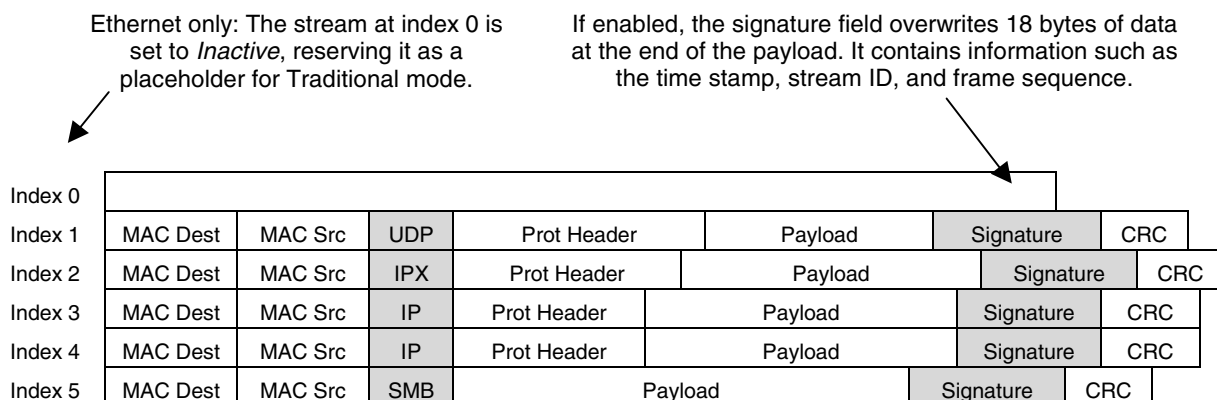
- Unique streams of traffic are generated from multiple frame blueprints.
- Information is tracked on a per-stream basis, as opposed to a per-port basis in the Traditional mode.
- There is a CRC check on the entire frame.

Additional features that apply to Ethernet and frame relay SmartMetrics are:

- The frame can contain embedded signature fields with information about each frame.
- There is in-depth latency and sequence information.
- IP streams contain an IP checksum.
- Traffic can be used to test Layer2, Layer3, and above.

The diagram below shows an example of SmartMetrics stream configuration for an Ethernet SmartCard. Note the multiple frame blueprints, different protocols, and varied frame sizes.

Ethernet and Frame Relay cards in SmartMetrics mode support the use of a signature field. This field contains information about the specific frame. The receiving card uses this information to analyze network traffic and provide histograms.



Support for Variable Field Definitions (VFDs)

A VFD (Variable Field Definition) is a field within the frame whose value can be manipulated—for example, incremented, decremented, or used in chunks. VFDs are written over existing information in the frame, such as the background fill pattern.

VFDs are available on Ethernet cards in the Traditional mode and on frame relay cards.

Ethernet The SmartMetrics mode does not support the use of VFDs, except that in the SmartBits customizable stream VFD3 can be used to enter the custom protocol header and payload.

Frame Relay The frame relay cards support VFD1 and VFD2 in SmartMetrics streams. They support VFD3 in limited fashion, where range is the total number of bytes usable from the VFD3 buffer.

Chapter 6:

Getting SmartMetrics Results

Histograms offer a powerful way to look at test results. Available in the SmartMetrics mode, they distill and analyze information about networks based on incoming test traffic. Whereas counters supply the cumulative numbers of events, histograms provide relational information and answer such questions as:

- How many frames were out of order and for which stream?
- At what points during the test did events take place?
- Which streams had latency issues and how often did they occur?

You can enable one histogram for each card each time a specific test is run. The selected histogram then analyzes the results of the current test traffic.

Histograms are available on the following SmartCards, SmartModules, and MiniModules: ML-7710, L3-6705, and L3-6710.

Five Histograms Types

For each test and card, you can enable one of the following five histogram types.

V2_LATENCY (Latency over Time)

This histogram provides the average, maximum, and minimum latency values for network traffic at specified intervals during the test. The values are the composite result of all streams averaged together. This histogram answers a question like: “*What was the over-all average latency in the first second?*”

You configure this histogram to analyze data at specified intervals. For example, you can specify latency measurements during the first nanosecond, second, third, and so forth.

V2_LATENCY_PER_STREAM (Combination Histogram)

This histogram gives an overall picture of latency per stream. It provides the minimum, maximum, and average latency values over the course of the test (Latency Per Stream). It tracks the occurrence of specific latency values on a per-stream basis (Latency Distribution). And it reports whether or not frames were received in sequence on a per-stream basis (Sequence Tracking).

LATENCY_DISTRIBUTION (Latency Distribution)

This histogram tracks the occurrence of specific latency values on a per-stream basis. It can show typical latency values and/or the distribution of multiple latency values.

You set the ranges of latency values using `Latency_Results_Settings`. A range of latency values can consist of consecutive values (such as in .4, .5, .6 microseconds) or varying values (such as .2, .5, 1.9...2.3 microseconds).

SEQUENCE (Sequence Tracking)

This histogram reports whether or not frames were received in sequence on a per-stream basis, and it identifies duplicate and dropped frames. Its algorithm is designed to match the operation of an actual TCP stack, and is as follows:

- As long as frames are received in sequence, the In Sequence counter is incremented.

- If a frame is received that is greater than the one expected, the number of missing frames (hole size) is noted, and a variable for the first of the missing frames is set.
- Subsequent in-order frames falling after the sequence hole increment the In Sequence variable.
- If the frame from the start of the hole is received, the hole-size variable is decremented.
- If a frame from the middle of the hole is received, the earlier frames still not received from the sequence hole are counted as Lost. The hole-size variable is decremented, and the start of the hole begins after the received frame. The expected frame continues to be one more than the last frame received in sequence.
- If another out-of-sequence frame is received before the previous sequence hole is filled, the Lost variable is incremented by the size of the previous sequence hole. The new hole is then tracked.
- If while the new sequence hole is being tracked, a previous out-of-sequence frame arrives, the Duplicate variable is incremented.
- The In Sequence value continues to increment for every frame received in sequence after the current sequence hole. For example:

1,2,3	Three frames in sequence.
1,2,3,9,10,11	Sequence hole five frames. 10,11, in sequence.
1,2,3,9,10,11,4	Sequence hole is now four frames.
1,2,3,9,10,11,4,15	First hole closed, Lost incremented by four. New hole three frames long.
1,2,3,9,10,11,4,15,5	Duplicate incremented by one. (5 is counted as a duplicate since the previous hole is no longer tracked).

RAW_TAGS (Bulk Data)

This histogram provides a list of statistics on a per-frame basis. It differs from other histogram results in that the data is not analyzed. Rather, Raw Tags gives you access to test data, so that you can analyze the information any way you wish. Because it generates records on a per-frame basis, Raw Tags creates a large number of records quickly.

How to Set up Histograms

Use the following steps to set up SmartMetrics histograms and retrieve results.

Step 1 Enable the Signature Field (Transmit Card)

Histograms depend on information in the Signature field. Signature fields are now supported by the L3 and frame relay cards, and they must be enabled when a stream is defined.

When creating frames with any of the following, enable the Signature field by setting

```
HTSetStructure
L3_DEFINE_n_STREAM
L3_DEFINE_MULTI_n_STREAM
L3_MOD_n_STREAM
FR_DEFINE_n_STREAM
FR_MOD_n_STREAM
ucTagField = 1
```

Step 2 Activate the Histogram (Receive Card)

Activate the desired histogram on the card using: HTSetCommand with L3_HIST_n or HTSetStructure with FR_HIST_n. The histogram is now in receive mode until you query it for information or records.

Message Function	iType1	Related Structure	Description
HTSetCommand	L3_HIST_<type>	See below	Activate the desired histogram.
HTSetStructure	FR_HIST_<type>	See below	Activate the desired histogram.

Histogram types include the following for L3.

Histogram	iType1 (Related Structure)
Latency over Time	L3_HIST_V2_LATENCY (Layer3HistLatency)
Latency per Stream	L3_HIST_V2_LATENCY_PER_STREAM (Layer3V2HistDistribution)
Latency Distribution	L3_HIST_LATENCY_DISTRIBUTION (Layer3HistDistribution)
Sequence Tracking	L3_HIST_SEQUENCE ()
Raw Tags	L3_HIST_RAW_TAGS ()

Histogram types include the following for frame relay (FR).

Histogram	iType1 (Related Structure)
Latency over Time	FR_HIST_V2_LATENCY ()
Latency per Stream	FR_HIST_V2_LATENCY_PER_STREAM ()
Latency Distribution	FR_HIST_LATENCY_DISTRIBUTION ()
Sequence Tracking	FR_HIST_SEQUENCE ()

Step 3 Clear Previous Records (Receive Card — Optional)

You can clear all previous records by using the HTSetCommand with L3_HIST_START. (There is no equivalent FR_HIST_START command.)

Message Function	iType1	Related Structure	Description
HTSetCommand	L3_HIST_START	None	Clear previous records. (There is no equivalent FR function.)

Step 4 Transmit Test Traffic (Transmit Card)

Start transmitting test traffic using a function such as HTRun or HGRun. You can send traffic however you wish (Step, Burst, or Constant stream). The port will continue to collect histogram data until histogram records or information is retrieved.

NOTE After you start transmitting, remember to allow enough time before attempting to retrieve histogram results.

Step 5 Query Card for Test Information (Receive Card — Optional)

You can find out how many records are on the card by using HTGetStructure with L3_HIST_ACTIVE_TEST_INFO or FR_HIST_ACTIVE_TEST_INFO. This information is useful, for example, when you do not want to retrieve all records.

Message Function	iType1	Related Structure	Description
HTGetStructure	L3_HIST_ACTIVE_TEST_INFO	L3HistActiveTest	Get the number of histogram records and the active histogram.
	FR_HIST_ACTIVE_TEST_INFO	None	

Step 6 Retrieve Histogram Results (Receive Card)

Once test traffic has been sent, retrieve histogram results using HTGetStructure with L3_HIST_n_INFO or FR_HIST_n_INFO.

Message Function	iType1	Related Structure	Description
HTGetStructure	L3_HIST_<type>_INFO	L3<type>Info	Get histogram results for active histogram <type>.
	FR_HIST_<type>_INFO	None	

The number of histogram records retrieved is determined by:

- The starting record specified by the index (iType2).
- The number of structures defined in pData.

The histogram records remain on the card until you clear them, select another histogram, or power off.

Once a histogram is enabled, it is in receive mode. It will continue to analyze all incoming frames containing signatures until it is queried for information or records.

You can stop the histogram receive process in two ways:

- Get histogram records using HTGetStructure with either L3_HIST_n_INFO or L3_AGGR_n_INFO.
- Get information about the histogram using L3_ACTIVE_TEST_INFO.

Chapter 7:

Working with Tcl

This chapter covers some basic Tcl concepts and offers notes specific to using Tcl with Netcom Systems' SmartLib programming library. (SmartLib supports a number of programming languages; this chapter focuses on working with Tcl.) For a complete, in-depth discussion of Tcl, consult one of the manuals specific to the language. A good source of information is the Scriptics website: www.scriptics.com.

Tcl is a *scripting language*. This means that you create a text-based *script* or application that is run from a Tcl shell. This is different from *compiled* languages like C or C++ that can create stand-alone, executable programs.

Tk is a graphic interface toolkit for Tcl. You can use it to create GUI interfaces that work with your SmartLib scripts.

One advantage of using Tcl is that you can send commands one at a time from the command line without compiling. For example, you can enter:

```
ETLink $ETCOM1
```

—and press <Enter> to link a computer to the SmartBits chassis; no procedure definition, no compilation. To create a script you use a basic text editor or programming editor, enter Tcl commands and commands from the SmartLib programming library, save it as text only, and run the script.

NOTE Tcl scripts must be saved as TEXT ONLY. If any other formatting is applied, it will corrupt your scripts.

Below you will find information and step-by-step instructions on:

- Setting up your environment.
- Running basic scripts.
- Using SmartLib's new improved Tcl interface.
- (Optional) Converting old scripts to use the new Tcl interface.
- General programming tips for using Tcl with SmartLib programming library.

Installing Tcl and SmartLib

Installation of SmartLib and Tcl is a simple, automated process. The SmartLib CD includes an installer for the Tcl scripting language and for the SmartLib library, including Tcl interface files to work with your version of Tcl.

To install both the current version of SmartLib and the Tcl scripting language, run the setup utility from the software CD

- For UNIX — `<your CD>/SmartLib/setup.sh`
- For Windows — `<your CD>\SmartLib\setup.exe`

The Setup installer will walk you step-by-step through the installation of both SmartLib and Tcl. If you already have Tcl on your computer, you can choose to only install the SmartLib programming library.

NOTE Remember the version of Tcl installed on your computer, so that you use the correct SmartLib files later.

For Windows users, it is suggested that you allow the installer to put the Tcl DLL in the Windows\System directory. This will ensure that the proper files are available when you run Tcl.

Setting up SmartLib with Tcl

You can set up your files and working directories in a number of ways, depending on your needs. This section explains which files you need and suggests *one possible configuration*.

Files You Will Need

To create Tcl scripts with SmartLib commands, you need to be able to run the Tcl shell, as well as load the proper files to make the SmartLib commands available. The tables below list the files you need. All are supplied on the installation CD.

SmartLib Files	File Use
et1000.tcl	The <i>old</i> SmartLib Tcl interface file. It contains the SmartLib commands that you use from your Tcl shell.
smartlib.tcl	The improved <i>new</i> SmartLib Tcl interface file. It is easier to use and contains the same SmartLib commands to use from your Tcl shell.
libetsmb.so (UNIX)	The main SmartLib file that must be loaded regardless of which programming language you use.
ETSMBW32.DLL (Win32)	
tclet100.so (UNIX)	The file that maps the Tcl interface file to the commands in the main SmartLib file.
TCLET100.DLL (Win32)	
misc.tcl	A Netcom System utility file used to display error/result messages when working with the Tcl shell.

Tcl Files	Tcl Version		File Use
	8.0	7.6	
tclsh80 (UNIX)	•		Runs the Tcl shell. This shell gives you a simple text-based window that you can type commands in.
tclsh76 (UNIX)		•	
tclsh80.exe (Win)	•		
tclsh76.exe (Win)		•	
tcl80.so (UNIX)	•		The main file used by the Tcl scripting language.
tcl76.so (UNIX)		•	
TCL80.DLL (Win)	•		
TCL76.DLL (Win)		•	
libtclStruct.so (UNIX)	•	•	A Tcl utility file used for creating structures. This utility must be used when working with Tcl and SmartLib.
TCLSTRUC.DLL (Win)	•	•	

Setting up Your Files

Some files depend on others to run. Here are four ways to ensure that files are available:

1. Put files in a shared, system directory, e.g. \Windows\System for Windows or /usr/bin for UNIX.
2. Copy dependent files into the same working directory.
3. Specify the path to the file in some sort of a configuration file, such as an *.ini file or init.tcl.
4. Put the full path into your path environment variable.

For this basic configuration, use a combination of the first two ways.

Step 1 All four *.DLL or *.so files (listed above) should be in the shared directory of your operating system. The SmartLib and Tcl installations can do that for you automatically. Two examples:

UNIX

```
[ your directory] /lib/libetsmb.so
                  /tcl100.so
                  /libtcl80.so
                  /libtclStruct.so
```

Windows

```
\ Windows\System\ETSMBW32.DLL
\ TCLET100.DLL
\ TCL80.DLL
\ TCLSTRUC.DLL
```

Step 2 All four *.DLL or *.so files (listed above) should be in the shared directory of your operating system. Put necessary files into a working directory. For example:

UNIX

```
/YourTclWork/smartlib.tcl
/misc.tcl
/tclsh80
/YourTclScripts.tcl
/et1000.tcl (for older scripts)
```

Windows

```
\YourTclWork\smartlib.tcl
\Misc.tcl
\tclsh80.exe
\YourTclScripts.tcl
```

Test Driving the Tcl Shell with SmartBits

If you followed the two previous sections, you have both SmartLib and Tcl installed on your computer, and the files you need are available from your working directory.

Your Hardware Setup

To make use of SmartLib, the computer must be hooked up to the SmartBits chassis.

NOTE See the *System Overview* manual in the *SmartBits User Guide* binder for a more extensive discussion of chassis setup.

Use both the Serial connection and the Ethernet connection on the back panel of the SmartBits. Use the Serial connection to configure the chassis's IP address. Once the IP address is set, you can use the Ethernet address to communicate to the chassis.

Setting the IP Address on the SmartBits

Step 1 Connect the computer COM Port to the chassis COM Port with the RS-232 straight-through cable supplied with your chassis. This cable is *not* a null-modem cable, although it has two male ends. Connect both the computer and the chassis to the same network.

NOTE Use the Ethernet port on the backplane of the chassis. The ports on the SmartCards can generate enough traffic to bring a network down.

Step 2 Set the IP address. Get an IP address from your Network Manager. Use a terminal-emulation software such as Hyper Terminal or Procomm Plus. Use the following commands to set the IP address:

```
ipaddr
ipaddr <new ip address>
ipaddr
```

Step 3 Close the terminal software and turn the chassis off and then on again. On a SmartBits 2000/200, the Link light on the chassis will blink on, then off again. This means the chassis is ready to be linked to the PC. On a SmartBits 6000/600, the chassis is ready to link when the Status light is green.

Use the Tcl Shell with a SmartLib Command

This section walks you through sending very basic SmartLib commands from the Tcl shell. (This example uses Tcl version 8.0.5.)

Step 1 **Run the tclsh80 shell.**

Go to your working directory (the one that contains Tclsh80 or Tclsh80.exe). Once you run the Tcl shell, you will see a text window with the % Tcl prompt.

NOTE For Windows-based PCs, Tcl version 8.0 and later allows you to use DOS commands such as **dir** and **cd** from the Tcl shell.

Step 2 Load the SmartLib Tcl interface file et1000.tcl.

To do this, at the % prompt type:

(For the *newer* interface) `source smartlib.tcl`

—or—

(For the *earlier* interface) `source et1000.tcl`

When these SmartLib commands are loaded, the message:

```
SmartLib Programming Library <version number>
```

—is displayed on the screen.

If this command is successful, the SmartLib commands are available and all needed DLLs or SOs have been loaded automatically. If this command fails, check the location of your files.

Step 3 Try a basic SmartLib command from the Tcl % prompt: Link the computer and the SmartBits chassis.

NOTE Tcl is case sensitive.

- If your computer to chassis connection is an Ethernet connection, configure the chassis IP address (different from the SmartCard IP addresses):

```
ETSocketLink nnn.nnn.nnn.nnn 16385
```

—where `nnn.nnn.nnn.nnn` is the chassis IP address and 16385 is the TCP port.

Press <Enter>. The green Link light on the chassis will light up when the connection is made.

- If, instead, you want to make a COM Port connection, use the appropriate COM Port number for your computer, and type:

```
ETLink $ETCOMn (where n is your COM Port)
```

Press <Enter>. The green Link light on the chassis will light up when the connection is made.

Step 4 Unlink.

Type:

```
ETUnLink
```

—and press <Enter>. The Link light will go off.

Step 5 Close the Tcl shell.

Type `exit` and press <Enter>.

Running a Sample Script

Once you have installed SmartLib and Tcl and established the link between your PC and the SmartBits chassis, you can run one of the many sample scripts provided on the CD.

SmartLib installs the code samples on your computer under the `Sample/Tcl` directory. Open them with a basic text editor and read the comments before you run the scripts.

The comments in the files will tell you:

- What tasks the script is accomplishing.

- What type of cards and which slots are used. (Many scripts demo a back-to-back connection between the first two cards in a chassis.)
- Which additional Tcl files you need to *source* before running the script. To source (run) a Tcl file, type `source FileName.tcl`.
- Whether you need to link first and source `et10000.tcl/smartlib.tcl`.
- If the script needs to call another configuration script (this is rarely done).

Step-by-Step: Running a Tcl Sample Script

Below is a step-by-step procedure to run one of the supplied Tcl sample scripts.

Step 1 Copy `ipstream.tcl`, `l3min.tcl`, and `startup.tcl` to your Tcl working directory.

These files can be found in:

```
./samples/tcl/smartlib_tcl\Layer3 (new interface)
./samples/tcl/et1000_tcl\Layer3 (older interface)
```

The `l3min.tcl` sample sends traffic from SmartMetrics Ethernet SmartCards. The `ipstream.tcl` sample configures the cards and is called by `l3min.tcl`.

Step 2 Make sure you have two SmartMetrics SmartCards in the first two slots of the SmartBits chassis (for example, ML-7710 cards in slot 0 and 1). Use a back-to-back cable to connect the two cards.

Step 3 Use a text editor to read the comments in `l3min.tcl` and `ipstream.tcl`. Then close the scripts.

NOTE Save scripts as *Text Only* if you make any changes.

Step 4 From your Tcl working directory, run the `Tclsh80` shell.

Step 5 At the Tcl command prompt (`%`), load the SmartLib Tcl commands. Type:

```
source smartlib.tcl
```

Step 6 Use SmartLib to establish a link between the computer and the SmartBits chassis. Use the `startup.tcl` script for this. Type:

```
source startup.tcl
```

Then follow the prompts to link. (You can also use the `ETLink` or `ETSocketLink` command).

Step 7 Run the script that transmits traffic. Enter:

```
source l3min.tcl
```

The file `l3min.tcl` will source `ipstream.tcl` to configure the traffic. You will see the traffic statistics on your monitor.

Step 8 When the script is done, you can run another script of your choice, or unlink using the `ETUnlink` command. Enter `exit` when you want to close the Tcl shell.

Using the New 3.07 Tcl Interface

The new Tcl interface is easier to use than previous library versions, while keeping the same library commands from previous versions. The Netcom Systems Library Team is dedicated to providing simpler development tools for faster development, while maintaining an easy transition between versions.

Backward Compatibility

Although the new Tcl interface supports simpler format and list commands, the old interface is also available for continued support of existing scripts. See *Converting Existing Scripts for the New Tcl* in this chapter if you wish to use old scripts with the simpler Tcl interface.

What Has Changed?

The Tcl focus for SmartLib has been *less typing, more results*. If you are using our Tcl interface, you will notice Tcl specific-benefits, as well as the new default values for all Message Function commands. Improvements include:

- Lists instead of arrays (new interface only).
- No special formatting for character fields, (new interface only).
- Default values for all Message Functions (both interfaces).
- Informative message when the library is sourced (both interfaces).
- Ability to send a single structure as an element of an array (both interfaces).

How Do I Run the New Tcl Interface?

SmartLib 3.07 now comes with two Tcl interface files:

- `et1000.tcl` Continued support for scripts written with SmartLib 3.06b and before.
- `smartlib.tcl` For scripts written with SmartLib 3.07 and later.

If you are creating new Tcl SmartLib scripts, source `smartlib.tcl` to use the improved Tcl interface.

If you are working with existing scripts and you don't want to convert them to work with `smartlib.tcl`, `et1000.tcl` will continue to support the older interface with no modifications needed.

NOTE The two interface files `et1000.tcl` and `smartlib.tcl` are *not compatible*. Remember to use the correct interface file with your scripts. See *Converting Existing Scripts for the New Tcl* in this chapter for more information.

Comparative Usage Examples

This section covers the Tcl improvements in detail. The `smartlib.tcl` examples are for 3.07 and later. The `et1000.tcl` examples support the earlier interface.

Lists Instead of Arrays (New Interface Only)

You can now use the native Tcl *lists* to send a list of numeric values with a SmartLib function. This is different from the earlier interface, where each value was sent as an element of an array.

smartlib.tcl example (new list of values):

```
set MyConfig(ucVFD1Data) {20 22 25 26 26 33 27 21}
```

et1000.tcl example (old array of values):

```
set MyConfig(ucVFD1Data.0.uc) [format %c 20]
set MyConfig(ucVFD1Data.1.uc) [format %c 22]
set MyConfig(ucVFD1Data.2.uc) [format %c 25]
set MyConfig(ucVFD1Data.3.uc) [format %c 26]
set MyConfig(ucVFD1Data.4.uc) [format %c 26]
set MyConfig(ucVFD1Data.5.uc) [format %c 33]
set MyConfig(ucVFD1Data.6.uc) [format %c 27]
set MyConfig(ucVFD1Data.7.uc) [format %c 21]
```

NOTE Arrays are still supported in smartlib.tcl. The format of SmartLib Tcl arrays has been simplified, however.

smartlib.tcl example (new array element format):

```
set MyConfig(ucVFD1Data.0) 20
```

et1000.tcl example (old array element format):

```
set MyConfig(ucVFD1Data.0.uc) [format %c 20]
```

No Special Formatting for Character Types (New Interface Only)

Characters are used extensively in the SmartLib Tcl interface to pass numeric values to the SmartCards. These values are now automatically recognized as numeric values (as opposed to ASCII characters) and require no additional formatting.

smartlib.tcl example (new passing the numeric value 20):

```
set MyConfig(ucVFD1Data.0) 20
```

et1000.tcl example (old passing a value using the Tcl formatting commands):

```
set MyConfig(ucVFD1Data.0.uc) [format %c 20]
```

Default Values

See *Default Values with the Tcl Interfaces* on page 45 for a complete discussion.

Informative Message When the Library is Sourced (Both Interfaces)

This is a convenient, although very minor, improvement in the Tcl interface. When the Tcl interface is sourced (smartlib.tcl loaded in memory) you will now see:

```
SmartLib Programming Library <version number>
```

—displayed. Previously, when the Tcl interface (et1000.tcl) was sourced (loaded in memory), the name of the last structure was displayed **PortPairStruct**.

Ability to Send One Array Element at a Time, When Arrays Are Used (Both Interfaces)

This improvement allows you to create an array of structures, then only send one member of the array. For example, you might use an array of structures to define multiple streams in L3_DEFINE_IP_STREAM.

Default Values with the Tcl Interfaces

Default Values for All Message Functions (Both Interfaces)

Default configuration values are an important new feature for versions of SmartLib 3.07 and later. These values are supported in both `smartlib.tcl` and `et1000.tcl`, as well as in all supported languages including Tcl, C/C++, VB, and Delphi.

The benefit of these values is that you send the configuration values you are interested in, and other default values are filled in for you.

NOTE Default values are covered in depth in the *SmartLib Message Functions* manual (see *Default Configuration Values*). They are also covered in this chapter because the additional automated defaults are available in the Tcl interface.

Automated Default Values

Automated default values are available in `smartlib.tcl` only. When you source `smartlib.tcl`, automated defaults are off by default. Use the **NSEnableAutoDefaults** command to make automated defaults available in your script. This command takes no parameters and is used once in the script, before the use of any default values. A related command, **NSDisableAutoDefaults**, turns off the automated defaults. (Both these commands are documented in the *SmartLib User Guide*.)

Default Value Examples

The default value interface is quite flexible. Below are the different ways you can use them from Tcl.

Sending Default Values without Defining a Structure (New *smartlib.tcl* Interface Only)

You can use the message function commands, using a dash ("-") to indicate the default structure. This will send all default values for the function.

Example

```
NSEnableAutoDefaults
```

```
HTSetStructure $GIG_STRUC_TX 0 0 0 -0 $iTxHub $iTxSlot $iTxPort
```

This dash means: "Use defaults. Don't store a structure."

Specifying Values without Specifying a Structure (New *smartlib.tcl* Interface Only)

You can use the dash (default values) and still specify certain values within the command.

Example

```
NSEnableAutoDefaults
```

```
HTSetStructure $GIG_STRUC_TX 0 0 0 -0 $iTxHub $iTxSlot $iTxPort \  
-ulGap 9600 \  
-ucVFD1Pattern {20 22 25 26 26 33 27 21} \  
-ucVFD2Pattern {30 32 35 36 36 43 37 31}
```

The dash is still used to specify default values without defining a structure. Three configuration values are specified. These values now override the defaults for Gap, VFD1, and VFD2.

Specifying a Structure Name So Configuration Values Can Be Accessed Later (New smartlib.tcl Interface Only)

You can specify a structure name so that your configuration values can be used later in your script. By putting the structure name as the fifth parameter of the Message Function, SmartLib will save your configuration values under the specified structure name.

Example

```
NSEnableAutoDefaults
HTSetStructure $GIG_STRUC_TX 0 0 0 MyConfig 0 $iTxHub $iTxSlot $iTxPort\
    -ulGap 9600 \
    -ucVFD1Pattern {20 22 25 26 26 33 27 21} \
    -ucVFD2Pattern {30 32 35 36 36 43 37 31}
```

The complete configuration including default values, gap, VFD1, and VFD2 will be saved in the automatically created structure: MyConfig.

Defining a Configuration Structure (with or without Default Values) So That Configuration Values Can Be Reused (Both smartlib.tcl and et1000.tcl Interfaces)

You can create a configuration structure and specify the values you are interested in. Other values will be filled in with defaults. The results are the same as if SmartLib automatically create a structure (as in the example above).

Example

```
struct_new MyConfig GIGTransmit
HTDefaultStructure $GIG_STRUC_TX MyConfig "" $iHub $iSlot $iPort
    set MyConfig(ulGap) 9600
    set MyConfig(ucVFD1Data) {20 22 25 26 26 33 27 21} \
    set MyConfig(ucVFD2Data) {30 32 35 36 36 43 37 31}
    HTSetStructure $GIG_STRUC_TX 0 0 0 MyConfig "" $iHub $iSlot $iPort
```

MyConfig is specified as the structure name.

Three of 16 values are specified.

MyConfig is used with the function call, and can be modified or called later in the script.

Sending Character Strings

The new interface allows you to send lists of numeric values. What if, within SmartLib or one of the SmartAPIs, you need to send a string?

To send a string within SmartLib, you can use this syntax to specify that it is a character string (different from a numeric list separated by spaces).

```
struct_new MyTestSetup TestSetup
set MyTestSetup(szLogFilename._char_) LogFileName.log
```

NOTE This is a tip for working with strings in either smartlib.tcl or et1000.tcl.

Converting Existing Scripts for the New Tcl

As mentioned earlier, both old and new Tcl interfaces are supported. However, if you have existing scripts, and wish to take advantage of the new, simplified Tcl interface, you will need to make some changes.

The paragraphs below describe possible cases when changes to existing scripts will be needed. They describe the changes a user *must* make to transform a script from old style to new style Tcl interface.

1. Use `smartlib.tcl` instead of `et1000.tcl`.

OLD

```
source et1000.tcl
```

NEW

```
source smartlib.tcl
```

2. Remove character formatting for characters. Any lines that perform character formatting need to be modified.

OLD

```
set gigtx(ucTransmitMode) [format %c $GIG_CONTINUOUS_MODE]
```

NEW

```
set gigtx(ucTransmitMode) $GIG_CONTINUOUS_MODE
```

3. Remove unnecessary type fields (`.c` and `.uc`) from array fields.

OLD

```
set gigtx(ucVFD1Data.0.uc) [format %c 0xAA]  
set gigtx(ucVFD1Data.1.uc) [format %c 0xAB]
```

NEW

```
set gigtx(ucVFD1Data.0) 0xAA  
set gigtx(ucVFD1Data.1) 0xAB
```

4. Add character casting for any structure fields that are character arrays representing strings.

OLD

```
set testsetup(szLogFilename.0.c) "a"  
set testsetup(szLogFilename.1.c) "."  
set testsetup(szLogFilename.2.c) "l"  
set testsetup(szLogFilename.3.c) "o"  
set testsetup(szLogFilename.4.c) "g"
```

NEW

```
set testsetup(szLogFilename._char_) "a.log"
```

5. Remove any `ConvertCtoI` commands (from `misc.tcl`) or any other use of the `scan` command which converts chars to ints.

OLD

```
puts "[ConvertCtoI $myCapData(ucData.0.uc)]"
```

NEW

```
puts "$myCapData(usData.0)"
```

Additional Information

This section goes into more detail about the SmartLib Tcl interface.

When SmartLib is Loaded

When SmartLib is loaded (either `smartlib.tcl` or `et1000.tcl`), it does the following:

1. Loads the interface library (`tclet100.dll` in Windows, `tclet100.so` in Unix). This library maps the Tcl SmartLib commands to their corresponding C/C++ SmartLib commands. (The interface library loads the C/C++ SmartLib: `etsmbw32.dll` in Windows, `libetsmb.so` in Unix.)
2. Loads the TclStruct 1.3 library. TclStruct is an extension to Tcl used to represent data structures in Tcl.
3. Initializes all predefined constants. All the `#define` statements in the C/C++ SmartLib header files have been translated to `set` statements in Tcl. This enables you to use these constants in your scripts.
4. Creates the SmartLib data structure types.

Error Checking

It is important to check for error conditions as your script is running. The utility file `misc.tcl` contains error-checking commands.

- `LIBCMD` Displays the function name, arguments, and return value if an error is received.
- `DCMD` Returns the function name, arguments, and any return value.

Example

```
LIBCMD HTGetCounters GigCounters $iRxHub $iRxSlot $iRxPort
```

Understanding Data Structures

Data structures are named groups of parameter values. With the new Tcl interface (`smartlib.tcl`), defining structures is optional. The older `et1000.tcl` interface required that you declare data structures.

All data structure types are declared in both `smartlib.tcl` and `et1000.tcl` using the `struct_typedef` command. To create a data structure using a declared structure from either interface file, use the `struct_new` command. For example, to create a data structure named `Mygt` of the type `GIGTransmit`, use the following syntax:

```
struct_new Mygt GIGTransmit
```

You can also create arrays of data structures by using the `struct_new` command. The following example creates the variable `MyStrms` to be an array of five `StreamIP` structures:

```
struct_new MyStrms StreamIP*5
```

Data structure fields are referenced by this syntax. The structure name is followed by an open parenthesis (followed by the name of the desired field to reference, followed by a close parenthesis). Sub-fields are separated by periods (.). For example, using the `MyStrms` variable created above, you could set the third stream's `uiFrameLength` field to 60 with:

```
set Mystrms(2.uiFrameLength) 60
```


Memory allocated for data structures using the `struct_new` command can be freed like any other variable in Tcl, by using the `unset` command. For example:

```
unset Mygt
unset MyStreams
```

Multi-Dimensional Arrays

Some SmartLib commands have multi-dimensional array arguments. Examples are `HTHubSlotPorts` and `HTCardModels`. SmartLib provides two utility functions, `ETMake2DArray` and `ETMake3DArray`, that create two- and three-dimensional arrays, respectively. The example below shows how to create and use a multi-dimensional array:

```
ETMake3DArray HSP $MAX_HUBS $MAX_SLOTS $MAX_PORTS
HTHubSlotPorts HSP
for {set iPort 0} {$iPort < $MAX_PORTS} {incr iPort} {
  for {set iHub 0} {$iHub < $MAX_HUBS} {incr iHub} {
    for {set iSlot 0} {$iSlot < $MAX_SLOTS} {incr iSlot} {
      puts $HSP($iHub,$iSlot,$iPort)
    }
  }
}
```

Pointers

In rare cases within SmartLib, a structure field may be a pointer to an array or structure. An example of this is the **Data** field of the `HTVFDStructure` structure. In C/C++ form, the `HTVFDStructure` is declared in `et1000.h` like this:

```
typedef struct
{
  int Configuration;
  int Range;
  int Offset;
  int* Data;    /*pointer to an array*/
  int DataCount;
} HTVFDStructure;
```

Since Tcl doesn't support pointers, we use another form of indirection. The `Data` field is declared as a character array instead. The Tcl structure as declared in `et1000.tcl` and `smartlib.tcl` is:

```
struct_typedef HTVFDStructure {struct
  {int Configuration}
  {int Range}
  {int Offset}
  {char*256 Data} #holds name of the array (up to 256 chars)
  {int DataCount}
}
```

The Data field is used to hold the name of an integer array created locally. The integer array is created as an array of Int structures:

```
struct_new mylocalData Int*50
```

For example purposes, let's say we have created a variable of type

HTVFDStructure:

```
struct_new myvfd HTVFDStructure
```

After filling in the local data array...

```
set mylocalData(0) 0xAA
set mylocalData(1) 0xAB
# ... etc ...
```

Set the Data field to be the name of the newly created integer array:

```
set myvfd(Data) mylocalData
```

Notice that there is no \$ in front of **mylocalData**. This is because the Data field is set to the actual string **mylocalData**, the name of the variable, not the value of that variable.

Sending Arrays

In some instances, you may want to pass multiple structures (an array of basic elements as the **pData** argument) when using HTSetStructure, HTGetStructure, and HTSetCommands. In these cases you must use an array of one of the single element utility structures: UChar, Char, Int, etc. Create an array of these structures and use that as the pData argument. The following example sets the background data to an incrementing pattern of 60 bytes:

OLD

```
struct_new mydata UChar*60
for {set i 0} {$i < 60} {incr i} {
    set mydata($i.uc) [format %c $i]
}
HTSetStructure $GIG_STRUC_FILL_PATTERN 0 0 0 mydata 0 $iHub $iSlot $iPort
```

NEW

```
struct_new mydata UChar*60
for {set i 0} {$i < 60} {incr i} {
    set mydata($i) $i
}
HTSetStructure $GIG_STRUC_FILL_PATTERN 0 0 0 mydata 0 $iHub $iSlot $iPort
```

Although you may use the **uiLen** argument to specify the size of the data being sent or received in the pData argument, it is not actually necessary to do so when using the Tcl interface (note the "" as the sixth parameter). The SmartLib Tcl interface calculates the size of the data being sent or received itself and passes this value on to the core SmartLib.

Chapter 8:

Programming in MS Windows™

This chapter provides information on programming in the Microsoft Windows™ environment. It includes installation instructions, directory and file definitions, general tips, and information specific to the C/C++, Tcl, Visual Basic, and Delphi compilers.

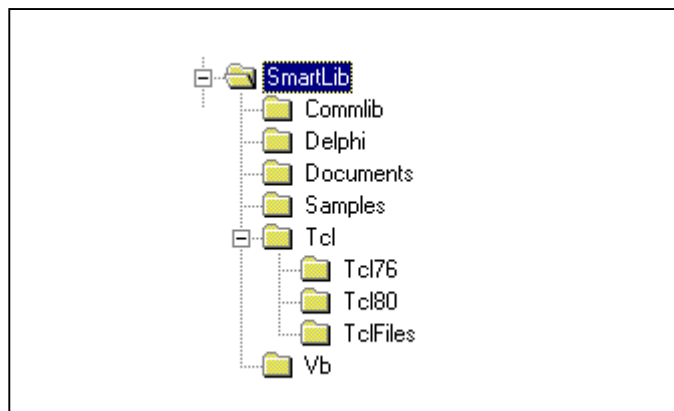
You can use this set of library functions to develop Microsoft Windows™-based applications on any IBM PC or compatible system that supports MS Windows.

SmartLib functions can be called from any program using the `cdecl` convention or the FAR PASCAL convention. Any MS Windows application capable of calling a Dynamic Link Library (such as Excel, National Instruments LabView, and Visual Basic) can use these functions.

Installation

AutoPlay for CDs automatically runs the SmartLib installation script when you put the CD into your PC. If *AutoPlay* is not enabled, run the `Setup.exe` from the root directory. Then follow the step-by-step instructions. SmartLib will be installed in a directory you choose.

The `Setup.exe` program creates the directory structure shown below.



These directories organize files by programming language. SmartLib provides multiple program interfaces with header and project files for each program environment. Complete source code comments can be found in the C/C++ files contained in the `CommLib` directory.

See the individual sections in this chapter for detailed information on the files for each programming language.

Directory Contents

The folders in the SmartLib directory have the following contents.

NOTE Sample code in Tcl, C, and VB can be found either on the CD or in your installation directory.

SmartLib

This directory (or the directory you selected for SmartLib) contains directories that hold program-specific files. In addition, it contains two files, `readme.html` and `license.pdf`.

Windows

This directory contains the `smartlib.dft` file, which holds default values for data structures. (See *Default Configuration Values* in the *SmartLib Message Functions* manual for further information.)

In addition, this directory contains subdirectories with SmartLib files for MS Windows systems, as listed below.

Commlib

This directory contains SmartLib's compiled DLL files for 16-bit and 32-bit Microsoft Windows. It also contains project and header files for C\C++. These contain the Original functions, the newer Message Functions, and the SmartAPI functions. This directory also contains legacy Visual Basic *.txt files (included for compatibility).

Delphi

This directory contains the source files needed to create SmartLib applications using the Delphi programming language.

Tcl

This directory contains subdirectories for Tcl files, as listed below.

Tcl\Tcl76

This directory contains the Tcl files for Tcl 7.6, including the SmartLib files `tcl76.dll`, `tclet100.dll`, `tclstruc.dll`, and an executable file needed to install Tcl 7.6, `win76p2.exe`.

Tcl\Tcl80

This directory contains the Tcl files for Tcl 8.0, including the SmartLib files `tcl80.dll`, `tclet100.dll`, and `tclstruc.dll`, and three files used to install Tcl 8.0, `cw3215.dll`, `tcl805.exe`, and `tcl80vc.lib`.

Tcl/TclFiles

This directory contains additional files for use with either version of Tcl.

VB

This directory contains SmartLib project and header files for Microsoft Visual Basic. These files include the 16-bit and 32-bit versions of the Visual Basic programming interface files. The directory also contains a file (`vb_tips.txt`) of tips on using SmartLib with Visual Basic.

General Programming Notes for Windows

The MS Windows link libraries are compiled with the Large memory model.

Import Library for 16-bit Applications

For MS Windows 16-bit applications, create an import library as follows:

1. Open a DOS box and go the directory where etsmbw16.dll is located.
2. Issue the command:

```
implib etsmbw16.lib etsmbw16.dll
```

The library will be created automatically.

NOTE SmartLib 3.07 is the last version to support 16-bit Windows.

Compatibility

Netcom Systems make every effort to keep SmartLib compatible with earlier versions. As more functions are added, you may need to re-link your application with the new library. For Microsoft Windows applications using the DLL, re-linking may not be necessary.

Developing with C/C++

For C/C++ program development, you must reference the ET1000.H file by using an `include` statement in your source files. This file provides the function prototypes, defined values, and structure declarations used by the library.

You must also link with the SmartLib *.LIB files that match your development environment.

Compiling Files

If you develop with Borland's C/C++, compile using SMBW32BC.LIB.

If you develop using Microsoft's C/C++, compile using SMBW32VC.LIB.

Applications from either compiler use the same SmartLib *.DLL during run time.

File Descriptions (Commlib Directory)

The files in the Commlib directory are used mainly when developing with C/C++, but this directory also contains SmartLib's central DLL files and legacy Visual Basic files.

File Name	Description
atmapi.h	In development for future release.
atmitems.h	Library header file of defines and structure definitions for ATM SmartCards.
atmitm32.txt	Visual Basic 5 legacy file needed only for compatibility with earlier Visual Basic/SmartLib applications.
atmsgapi.h	Library header file of the Smart API for ATM Signaling tests.
et1000.h	Library header file of basic defines, structure definitions, and all function prototypes.
ethitems.h	Library header file of defines and structure definitions for the new Ethernet Message Functions.
etsmb16A.dll	<i>For Netcom Systems internal use.</i>

File Name	Description
etsmb16C.dll	<i>For Netcom Systems internal use.</i>
etsmb16R.dll	<i>For Netcom Systems internal use.</i>
etsmb16T.dll	<i>For Netcom Systems internal use.</i>
etsmb16V.dll	<i>For Netcom Systems internal use.</i>
etsmbapi.txt	Visual Basic 3 legacy file needed only for compatibility with earlier Visual Basic/SmartLib applications.
Etsmbw16.dll	Dynamic link library for use with 16-bit applications developed for Windows 95 or NT.
Etsmbw32.dll	Dynamic link library for use with 32-bit applications developed for Windows 95 or NT.
etsmbw32.txt	Visual Basic 5 legacy file needed only for compatibility with earlier Visual Basic/SmartLib applications.
ettypes.h	Library header file of necessary ETSMB variable types (such as U64 when working with 64-bit numbers).
frame.h	Library header file for the newer, easier frame-building functions: NSCreateFrame, NSSetPayLoad, HTFrame, NSDeleteFrame, NSCreateFrameAndPayLoad, NSModifyFrame.
fritems.h	Library header file of defines and structure definitions for the Frame Relay cards.
fstitems.h	Library header file of defines and structure definitions for the Fast Ethernet (100MB) cards.
gigitems.h	Library header file of defines and structure definitions for Gigabit Ethernet cards.
l2items.h	Library header file of defines and structure definitions for Layer 2 cards.
l3items.h	Library header file of defines and structure definitions for Layer 3 and Multi-layer cards.
positems.h	Library header file of defines and structure definitions for POS cards.
pppitems.h	Library header file of defines and structure definitions for PPP functions.
smbw32bc.lib	The Borland C/C++ compatible import library used with the ETSMBW32.DLL for 32-bit applications.
smbw32vc.lib	The Visual C/C++ compatible import library used with the ETSMBW32.DLL for 32-bit applications.
stmitems.h	Library header file of defines and structure definitions for some common Stream items.
tcpisp.h	In development for future release.
tcpitems.h	In development for future release.
testapi.h	Library header file of the Smart API for RFC-1242 and RFC-1944 Tests.
testcmmn.h	Library header file of common defines and structure definitions for the Smart APIs.
wanitems.h	Library header file of defines and structure definitions common to both Wide Area Network SmartCards (ATM and Frame Relay). This file includes defines such as DSI, EI, and DS3.

Developing with Tcl

Tcl is a flexible programming language, noted for its on-the-fly command-line capabilities. Tcl enables you to test a function call from the text-based command line without having to compile a program. This allows you to test your code line by line.

SmartLib supports Tcl Versions 7.6 and Tcl 8.0 in both Windows and UNIX. Both Tcl versions are included with the SmartLib Software Developer's Kit, along with the SmartLib files needed to develop test applications with Tcl.

Related Information

For an extensive discussion on using SmartLib with Tcl, see *Working with Tcl* in this manual. For Tcl examples, see the files under \Samples\Tcl\ on the CD or your installation drive.

Tcl Directory

The Tcl directory contains three subdirectories, as described below.

Tcl/Tcl76 Directory

The Tcl76 directory contains following files.

File Name	Description
tcl76.dll	Tcl project library used when creating applications.
tclstruc.dll	Tcl DLL used to create structures.
tclet100.dll	SmartLib API for Tcl. This file maps Tcl calls to the main ETSMB*.DLL.
win76p2.exe	Executable file for installing Tcl 7.6.

Tcl/Tcl80 Directory

The Tcl directory contains the following files.

File Name	Description
tcl80.dll	Tcl project library, used when creating applications.
tclstruc.dll	Tcl DLL used for creating structures.
tclet100.dll	SmartLib API for Tcl. This file maps Tcl calls to the main ETSMB*.DLL.
cw3215.dll	DLL for installation of Tcl 8.0.
tcl805.exe	Executable file for installing Tcl 8.0.
tcl80vc.lib	Library file to be used with Tcl 8.0p5.

Tcl/TclFiles Directory

The TclFiles directory contains the following files.

File Name	Description
misc.tcl	Tcl error-handling utility used to capture return values.
show.tcl	Tcl Utility used to view elements in a structure.
et1000.tcl	SmartLib header file containing SmartLib <code>defines</code> , structure definitions, and function prototypes for the original SmartLib Tcl interface.
smartlib.tcl	SmartLib header file for the new SmartLib Tcl interface.

Developing with Delphi

File Descriptions

The required interface files are in the DELPHI directory. Each *.PAS file corresponds to a C/C++ header file (.h file). For file descriptions, see page 53, *File Descriptions (Commlib Directory)*.

The central SmartLib DLL is located in the Commlib directory.

Developing with Visual Basic

The SmartLib Programming Library includes files for the Microsoft Visual Basic environment. Much of the information for C/C++ also applies to Visual Basic. Exceptions and differences are noted in this section.

Important Differences between Visual Basic and C/C++

Case Sensitivity and Parameter Names

C/C++ is case sensitive; Visual Basic is not. Some parameters with identical functions have different names in each language, as shown below.

C/C++	Visual Basic (SmartLib Previous)	Visual Basic (SmartLib 3.02 and Later)
HTSTOP	HTRUN_STOP	Use either name.
HTSTEP	HTRUN_STEP	Use either name.
HTRUN	HTRUN_RUN	HTRUN_RUN or HTRUN_VALUE NOTE: Applies only to constant parameter. Do not change the name of the HTRUN function.
ETSTOP	ETRUN_STOP	Use either name.
ETSTEP	ETRUN_STEP	Use either name.
ETRUN	ETRUN_RUN	Use either name.

Space for Integers

In Visual Basic, integers require the same amount of space in both the 16-bit and 32-bit versions. In C/C++, `int` requires a larger memory allocation in the 32-bit version than in the 16-bit version. This means that items appearing in this manual as `int` are declared as `Long` in the SmartLib header and LIB files for 32-bit Visual Basic.

Unsigned Types

Visual Basic does not support unsigned types. In some cases where unsigned types are specified, conversions must be made. An example is a counter result where all 32 bits are used to represent a positive number.

Parameters for the HTVFDStructure

VB parameter names for the HTVFDStructure now match more closely the parameter names used in C/C++.

C/C++	Visual Basic (SmartLib Previous)	Visual Basic (SmartLib 3.02 and Later)
*Data	iPointer	pData
DataCount	iLength	DataCount

Files for Visual Basic

The interface files needed to use SmartLib with Visual Basic are in the Vb directory. DLLs and legacy files are in the Commlib directory. Each *.B16 or *.B32 file corresponds to a C/C++ header file.

For file descriptions, see page 53, File Descriptions (Commlib Directory).

The following files are used when developing with Visual Basic. To use the SmartLib functions, data structures, and constants, include the appropriate *.b16 or *.b32 file in your VB project.

File Name	Description
ETSMBW16.DLL	Dynamic link library for use with 16-bit applications developed for Windows 95 or NT. This file is located in the Commlib directory and installed in your Windows\System directory.
ETSMBW32.DLL	Dynamic link library for use with 32-bit applications developed for Windows 95 or NT. This file is located in the Commlib directory and installed in your Windows\System directory.
*.B16	Library header files of defines, structure definitions, and function prototypes. These files are used for VB 16-bit.
*.B32	Library header files of defines, structure definitions, and function prototypes. These files are used for VB 32-bit.
ETSMBAPI.TXT	Visual Basic legacy files located in the Commlib directory.
ETSMBW32.TXT	Visual Basic legacy files located in the Commlib directory.
ATMITM32.TXT	Visual Basic legacy files located in the Commlib directory.

Chapter 9:

Programming in UNIX

This chapter provides information on programming in the UNIX environment. It includes installation instructions, directory and file definitions, general tips, and information specific to the C/C++ and Tcl compilers.

SmartLib supports C and Tcl (Versions 7.6 and 8.0) for the UNIX programming environment. It includes extensive Tcl and C code examples.

SmartLib has been tested under the UNIX versions listed below:

- SunOS 4.1.4.
- Solaris 2.5.1 (SPARC architecture).
- Solaris 2.5.1 (x86 architecture).
- Red Hat Linux version 5.2 and above (x86 architecture).

Installation

To install SmartLib under UNIX, run the `setup.sh` installation utility and pick the files you wish to install. The CD contains both source code and pre-compiled shared libraries.

Use the following steps.

NOTE Before you install SmartLib, the following programs must be installed on your system and in your PATH: `gcc` (including the standard C++ library), `make`, and `gunzip`.

1. Insert the SmartLib CD-ROM into your CD drive.
2. Mount the CD. This works differently on different platforms:
 - ❖ Under Solaris, it is automatic. Your CD will be mounted at `/cdrom/netcom`.
 - ❖ Under Linux, enter `mount -r /dev/cdrom /mnt/cdrom`. Your CD will be mounted at `/mnt/cdrom`.
 - ❖ Under SunOS, use the correct mount command.
3. Change to the directory where the CD is mounted.
4. Run the Setup script `setup.sh`. You will be prompted on how to customize the SmartLib installation to meet your needs. Important questions include:

Where should SmartLib files be installed?

Several subdirectories are created, depending on which features you choose to install. For system-wide access, it is best to install as `root` and place SmartLib in `/usr/local`. If you don't have root access, you can install in your account. For example, if your home directory is `/export/home/jdoe`, enter the following:

```
/export/home/jdoe/smartlib
```

Do I want precompiled versions of SmartLib, or do I want to compile the source files on my system?

In most cases, use the precompiled versions. They have been tested and will install much faster. If you elect not to install the precompiled version, the source files are installed instead then compiled in your environment during the install process.

When installing with Linux, `libc.so` and `libn.so` may have been renamed so that our installation script cannot find them. To correct this problem, create a symbolic link (a small pointer file) in the directory where you would like `libc.so.n` and `libn.so.n` to reside. An example is shown below.

```
ln -s libc.so libc.so.5
ln -s libn.so libn.so.5
```

Will I write scripts with Tcl?

If so, which version, 7.6 or 8.0? You can also install the Tcl programming language that is provided on the CD.

UNIX Directory Structure and Content

The following are all possible directories that can be created when installing SmartLib under UNIX. In practice, a subset of these will be loaded on your computer, depending on your selections during installation. This directory structure can be expanded. The list describes the top-level directories.

/bin

Contains files used to run the Tcl shell (tclsh is a pointer to the current Tcl shell file).

/include

Contains C header files and Tcl files used when coding with SmartLib.

/lib

Contains the compiled *.so files. This directory may include *.so files for Tcl if the Tcl interface was selected.

/lib/tcl8.0

Contains Tcl 8.0 programming files, if the 8.0 Tcl compiler was installed.

/lib/tcl7.6

Contains Tcl 7.6 programming files, if the 7.6 Tcl compiler was installed.

/man

If Tcl is installed, numerous Tcl topics are added to the /man/* directories.

The following files are removed after a successful installation:

/tmp

Contains other directories used if source code is compiled on the PC, instead of installing precompiled files. Once SmartLib and/or Tcl library files are compiled, this directory can be deleted.

/tmp/proglib

Contains SmartLib's C source files used to compile the main SmartLib *.os file libetsmb.so. This file supports the Original Functions, Message Functions, and the SmartAPI functions.

tmp/Tcl8.0.5

Contains files and subdirectories used to install Tcl 8.0p5.

tmp/Tcl7.6

Contains files and subdirectories used to install Tcl 7.6.

tmp/tclstruct

Contains Tcl files used when compiling libtclstruct.so. Once compiled, this file is used to work with structures in Tcl. It is stored in the /lib directory. It must be included when working with SmartLib in Tcl.

tmp/tclext

Contains Tcl files used when compiling tclet100.so. This file is the Tcl interface to the C function calls. Once compiled it is stored in the /lib directory. It must be included when working with SmartLib in Tcl.

tmp/tcl

More temp files.

Developing with C/C++

For information and file descriptions specific to the SmartLib C/C++ interface, see *Developing with C/C++* (page 53).

Developing with Tcl

Basic Programming Information

For file descriptions specific to the Tcl interface, see *Developing with Tcl* (page 55).

Step-by-step Information

For an in-depth discussion of using SmartLib with Tcl, see Chapter 7, *Working with Tcl* (page 37).

Code Samples

For Tcl examples, see the files under `Samples\Tcl\` on the CD or on your installation drive.

Chapter 10: Code Examples

SmartLib provides extensive source code examples in both C++ and Tcl, to guide you through the basic tasks in working with SmartLib.

- The Tcl demo scripts contain well-commented code that is used both in training and in the field.
- The C examples walk you through a series of basic tasks while configuring different cards for Traditional and SmartMetrics traffic.
- A Visual Basic (VB) example shows basic test procedures when using Microsoft Visual Basic.

We recommend that you look at the Tcl examples regardless of your programming environment.

Cross-reference to Functions and Tcl Examples

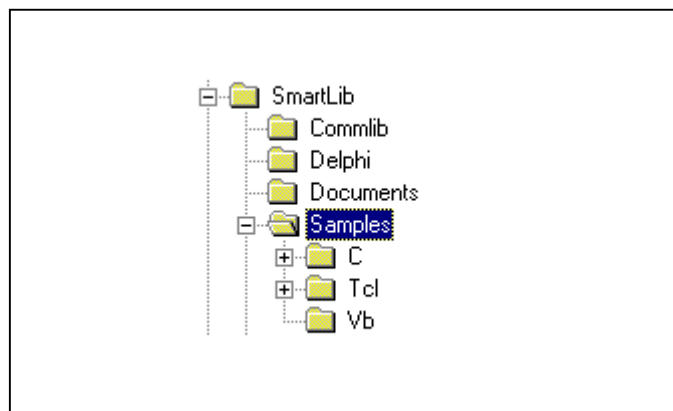
Each coding example illustrates the use of one or more Original Functions or Message Functions. To help you find instances of function use in the Tcl examples, the tables that follow list each coding example, describe its purpose, and also cite the functions that it demonstrates.

NOTE Many Tcl coding examples include the same commonly used functions—for example, to link the PC to the SmartBits, group cards for testing, and start and stop tests. These common functions are included in each instance, but the functions that are uniquely demonstrate by a coding example are shown in **boldface** type.

At the end of this chapter, the table *Cross-reference to Functions and Coding Examples* lists all the Original Functions and Message Functions that appear in the Tcl coding examples and identifies the example(s) that contain them.

Where to Find the Code Examples

The SmartLib examples are located on the C D in the following directories.



Tcl Demo Scripts

The Tcl Demo scripts cover key tasks in SmartLib. They have been created, refined, and used by our Technical Support Specialists and offer practical answers to questions received from customers. The scripts are heavily commented and contain information useful to SmartLib programmers working in any environment.

The tables below summarize the scripts available for the different card families. For step-by-step instructions on working with the Tcl scripts, see *Working with Tcl* in this manual.

NOTE: SmartLib 3.07 contains Tcl examples for both the older Tcl interface (et1000.tcl) and the new improved Tcl interface (smartlib.tcl). If you do not find an example in the smartlib_tcl directory, you can check in the et1000_tcl directory and then make basic modifications needed for the new Tcl interface. You can also just source the older interface file.

Examples in the et1000_tcl Directory

All Cards (AllCards)

These scripts show basic, preliminary tasks executed by SmartLib.

File Name	Description	Demonstrates
1stlink.tcl	A simple routine to link between the PC serial port and a Smartbits controller.	ETLink / ETUnLink
Backoff.tcl	Sets the backoff time (how quickly an Ethernet card attempts to transmit after a collision).	HGClearGroup HGAddtGroup HGResetPort HGCollisionBackoff Aggressiveness HGStart / HGStop
cardmod.tcl	Returns the card model. Example of a Tcl two-dimensional array.	ETMake2DArray HTCardModels
gap.tcl	Sets the Interframe Gap, decrementing the gap with each code loop.	HTCountStructure HTRResetPort HTDuplexMode HTTransmitMode HTDataLength HTBurstCount HTGap HTRun HTGetCounters
GapPerCent.tcl	Calculates the % utilization and sets the Tx parameters at that rate. (This sample code does not work with ATM and WAN cards.)	HTDataLength HTSetSpeed HTGap HTClearPort HTCountStructure HTRun HTGetCounters
Group.tcl	Creates a group of two cards and transmits traffic.	HGClearGroup HGAddtoGroup HGResetPort HGStart HGStop

File Name	Description	Demonstrates
GroupCount.tcl	Creates a group of two cards, transmits traffic, and retrieves and displays group counter information.	HGSetGroup HGAddtoGroup HGStart HGStop HGGetCounters
LibVer.tcl	Example of passing strings in Tcl. Gets the SmartLib version.	ETGetLibVersion
multi-link.tcl	Demonstrates the use of multiple simultaneous links to more than one SmartBits chassis.	ETLink / ETUnLink HTRun / HTStop ETSetCurrentLink
ping_l2.tcl	Sets up PING packets on a Layer 2 card.	HTResetPort HTFillPattern HTVFD HTDataLength HTTransmitMode HTBurstCount HTGapAndScale HTRun
SocketLink.tcl	A simple routine for an Ethernet link between the PC and a Smartbits controller.	ETSocketLink / ETUnLink
Startup.tcl	Sample code to include at the beginning of a Tcl script. It checks to see if the Netcom Systems programming library (ET1000.TCL) has been sourced.	ETGetLibVersion
Trigger.tcl	Sets a group of two cards (slot 1 and 2), then uses HGIsHubSlotPortInGroup to test card membership. Sets up a VFD and a trigger to match. (This sample code does not work with ATM and WAN cards.)	HGSetGroup HGAddtoGroup HGIsHubSlotPortInGroup HGVFD HTTrigger HGGetCounters
vfd.tcl	Demonstrates use of VFDs and shows the differences between the VFDs. Creates traffic with VFD1, VFD2, and VFD3.	HTFillPattern HTVFD

ATM (ATMCard)

These scripts work with the ATM SmartCards.

File Name	Description	Demonstrates
Atm_ilmi.TCL	Allows an ATM Card to register its 20-byte address with the network device (ATM switch) it is connected to. After successful registration, it displays the ILMI Status information.	HTSetCommand <i>with:</i> ATM_ILMI_REGISTER ATM_ILMI_DEREGISTER HTGetStructure <i>with:</i> ATM_ILMI_INFO
ATM_SAAL.tcl	Establishes management connections with an ATM switch.	HTSetCommand <i>with:</i> ATM_SAAL_ESTABLISH HTGetStructure <i>with:</i> ATM_SAAL_INFO

File Name	Description	Demonstrates
ATM_SetCount.tcl	Removes all streams from cards, then sets up a range of PVC streams and starts transmitting. A back-to-back test, it ensures that the number of frames received equals the number of frames transmitted.	HTGetStructure with: ATM_CARD_CAPABILITY ATM_STREAM_DETAIL ATM_STREAM_DETAIL_INFO ATM_VCC_INFO HTSetStructure with: ATM_STREAM_CONTROL ATM_STREAM ATM_FRAME_DEF
ATM_Status.tcl	Uses HTGetEnhancedStatus and HTGetLED Original Functions to checks the status and LED state of an ATM card.	HTGetEnhancedStatus HTGetLED
atm_trig.tcl	Uses ATM_Set&Count.tcl as a base, but adds triggers and uses a different mechanism to retrieve individual VCC frame counts.	HTGetStructure with: ATM_CARD_CAPABILITY ATM_STREAM_DETAIL_INFO ATM_VCC_INFO ATM_CONN_TRIGGER_INFO HTSetStructure with ATM_STREAM_CONTROL ATM_STREAM
ATMClip.tcl	Shows Classical PVC to PVC connections between two cards; creates two PVC streams. This demo requires conection to a device with a CLIP server if you use the CLIP server commands.	HTGetStructure with: ATM_CARD_CAPABILITY HTSetStructure with: ATM_STREAM ATM_FRAME_DEF ATM_STREAM_CONTROL HTRun
ATMData.tcl	Retrieves and displays the settings and capabilities of an ATM card.	HTGetStructure with: ATM_CARD_TYPE ATM_CARD_INFO ATM_CARD_CAPABILITY
ATMGetCounts.tcl	Simple PVC to PVC connection between two cards. Shows use of enhanced stream indexes by using the newer ATM_STREAM_VCC_INFO command to get per-stream data, without needing to get the connection index first.	HTGetStructure with: ATM_CARD_CAPABILITY HTSetStructure with: ATM_STREAM_CONTROL ATM_FRAME_DEF HGSetGroup HGAddtoGroup HGClearPort HGStart / HGStop
ATMSonetInfo.tcl	Retrieves and displays SONET Section / Line / Path error information.	HTGetStructure with: ATM_SONET_INFO

File Name	Description	Demonstrates
pppdemo.tcl	Tests PPP frames encapsulated over ATM using LLC Encapsulation per RFC-1483, "Multiprotocol Encapsulation over ATM AAL-5", or the VC-based multiplexing technique per RFC-2364, "PPP over AAL-5."	HTGetStructure <i>with</i> : ATM_CARD_CAPABILITY ATM_STREAM_DETAIL_INFO PPP_STATUS_INFO HTSetStructure <i>with</i> : ATM_STREAM_CONTROL ATM_LINE ATM_STREAM ATM_STREAM_PARAMS_COPY ATM_STREAM_PARAMS_FILL PPP_PARAMS_COPY PPP_PARAMS_FILL PPP_SET_CTRL ATM_STREAM_VCC_INFO HGClearGroup HGAddtoGroup HGClearPort HGStart / HGStop

ET-1000 (et1000)

These scripts deal with ET-1000 functionality. The ET-1000 is a precursor to the SMB 1000 with two ports and SmartCards that are not removable.

The ET-1000 examples include code for an ET-1000 as well as for ST-64XX cards emulating an ET-1000. This functionality can be useful when, for example, you have ST-6410 SmartCards and want to capture test frames.

The ET functions listed below are documented in Appendix A, *Original Functions for the ET-1000 Only* (see page 197).

File Name	Description	Demonstrates
ET1000MODE.tcl	Defines frames with VFDs and then transmits traffic. These routines use ST-64XX cards and a SmartBits chassis to access ET-1000 functionality.	HTSelectTransmit HTSelectReceive ETDataLength ETDataPattern ETVFDParams ETSetSel ETVFDRun ETRun
ETVFD_CYCLE.TCL	Defines frames with VFDs and then transmits traffic. These routines execute the same functions as ET1000Mode.tcl (above) but control an actual ET-1000.	<i>Same as above plus:</i> ETBurst
multi.tcl	General overview of ET-1000 capabilities.	<i>Same as above plus:</i> ETReceiveTrigger ETMFCounter ETCaptureParams ETGetCapturePacket

Ethernet (ETH)

These scripts work 10Mbps or 10/100Mbps Ethernet cards and 10/100 Layer 3 cards. For completeness, the properties for all cards are presented; however, running against non-Ethernet cards will return an error code from the function.

File Name	Description	Demonstrates
ETHCardInfo.tcl	Demonstrates use of the ETH_CARD_INFO message function. Allows use of legacy Ethernet card functions with HTGetStructure Message Function.	HTGetStructure with: ETH_CARD_INFO
ETHTransmit.tcl	Demonstrates how to set up test frames then transmit.	HTSetStructure with: ETH_TRANSMIT

Fast Ethernet (FastCard)

These scripts work with the SX-7210 and SX7410 Fast Ethernet SmartCards. These cards support 10/100 Mbps traffic. They do not support histograms and VTEs; that is, there is no signature field.

File Name	Description	Demonstrates
capture.tcl	Configures a main traffic stream as well as an alternate stream (e.g., an error stream), transmits traffic, then captures incoming traffic and displays the capture.	HTTransmitMode HTBurstCount HTDataLength HTFillPattern HTSetStructure with: FST_ALTERNATE_TX FST_CAPTURE_PARAMS FST_CAPTURE_COUNT_INFO
Collision.tcl	Demonstrates how to set up collision generation on SX-74xx and SX-72xx cards.	HTCollision HGSetGroup HGAddtoGroup HGDuplexMode HGClearPort HTTransmitMode HTBurstCount HTRun HGGetCounters
EnhancedStats.tcl	Uses HTGetCardModel to ensure a FastCard is selected, then does a bitwise AND to see if the FAST7410_STATUS_LINK (0x0000200h) is set.	HTGetCardModel HTGetEnhancedStatus
FastTrig&VFD.tcl	Shows how to set up VFDs and trigger on a pattern, then get and display counter data.	HTResetPort HGSetGroup HGAddtoGroup HGSetSpeed HTTransmitMode HTBurstCount HTFillPattern HTTrigger HTGetCounters

File Name	Description	Demonstrates
gap.tcl	Illustrates setting gap, speed, and duplex mode.	HTResetPort HTSetSpeed HTDuplexMode HTTransmitMode HTDataLength HTGap HTRun HTGetcounters
mii.tcl	Demonstrates the use of the HTReadMII and HTWriteMII commands to enable Autonegotiation and to display the contents of the MII registers and force Autonegotiation.	HTResetPort HTFindMIIAddress HTReadMII HTWriteMII
setspeed.tcl	Uses the HT commands to set speed, mode, and duplex mode for individual cards and for groups of cards.	HTResetPort HTSetSpeed HTDuplexMode HTTransmitMode HTDataLength HTRun HGClearGroup HGAddtoGroup HGResetPort HGSet Speed HGDuplexMode HGTransmitMode HGDataLength HGStart / HGStop

Gigabit Ethernet (GIG)

These scripts work with the Gigabit Ethernet SmartCards and SmartModules, including the GX-1405, GX-1405B, LAN-6200A and LAN-3200A.

File Name	Description	Demonstrates
GIGCount.tcl	Shows how to create a group of Gigabit cards, define an array of counter structures, then draw a simple report and fill in counter data.	HGSetGroup HGAddtoGroup HTSetStructure <i>with:</i> GIG_STRUC_TX HGClearPort HGStart / HGStop HGGetCounters
GIGVFD.tcl	Basic demonstration of how to start, transmit, capture, and display data with Gigabit cards.	HTSetStructure <i>with:</i> GIG_STRUC_FILL_PATTERN GIG_STRUC_VFD3 GIG_STRUC_CAPTURE_SETUP GIG_STRUC_CAP_DATA_INFO HTRun

Layer 3 (Layer3)

These scripts illustrate how to create streams with SmartMetrics SmartCards such as the L3-6710 and the ML-7710.

File Name	Description	Demonstrates
ipstream.tcl	Creates multiple IP streams.	HTSetStructure with L3_DEFINE_IP_STREAM L3_DEFINE_MULTI_IP_STREAM
ipxstream.tcl	Creates multiple IPX streams.	HTSetStructure with L3_DEFINE_IPX_STREAM L3_DEFINE_MULTI_IPX_STREAM
L2-L3.tcl	Shows how to switch the ML-7710 SmartCard from Traditional mode to SmartMetrics mode.	HTGetStructure with L3_CAPTURE_PACKET_DATA_INFO HTSetStructure with L3_DEFINE_IP_STREAM HTVFD HTTransmitMode HTBurstCount HTRun
l3_hist_raw_tags.tcl	Sets up a series of streams externally (by sourcing ipstream.tcl), then transmits a burst of packets and displays the distribution.	HTGetStructure with L3_DEFINE_STREAM_COUNT_INFO L3_HIST_RAW_TAGS_INFO HTSetCommand with L3_HIST_RAW_TAGS HTTransmitMode HTBurstCount HTRun
L3_HIST_V2_LAT_PER_ST.tcl	Sets up a series of streams externally (by sourcing ipstream.tcl) then transmits a burst of packets and displays the distribution.	HTGetStructure with L3_DEFINE_STREAM_COUNT_INFO L3_HIST_RAW_TAGS_INFO HTSetCommand with L3_HIST_RAW_TAGS HTTransmitMode HTBurstCount HTRun
l3_SB.tcl	Layer 3 script that calls SBstream.tcl to set up Layer 3 cards and create streams, then transmits for three seconds and displays number of frames.	HTSetCommand with L3_START_ARPS L3_CAPTURE_OFF_TYPE L3_CAPTURE_ALL_TYPE HTGetStructure with L3_CAPTURE_COUNT_INFO L3_CAPTURE_PACKET_DATA_INFO HTTransmitMode HTBurstCount HTDataLength HTFillPattern HTRun
L3_STREAM_INFO.tcl	Sets up one IP stream on an ML-7710 (Layer 3) card, then reads and displays card data.	HTSetStructure with L3_DEFINE_IP_STREAM HTGetStructure with L3_DEFINED_STREAM_COUNT_INFO L3_STREAM_INFO

File Name	Description	Demonstrates
L3_V2_HIST_LAT.tcl	Sets up a series of streams externally by sourcing ipstream.tcl, then transmits a burst of packets and displays the distribution.	HTGetStructure with L3_DEFINED_STREAM_COUNT_INFO L3_HIST_V2_LATENCY_INFO HTSetCommand with L3_HIST_V2_LATENCY HTTransmitMode HTBurstCount HTRun
l3min.tcl	Performs a minimum configuration for a SmartMetrics card, sets up traffic streams, runs, and displays number of frames.	HTSetCommand with L3_START_ARPS L3_CAPTURE_OFF_TYPE L3_CAPTURE_ALL_TYPE HTGetStructure with L3_CAPTURE_COUNT_INFO L3_CAPTURE_PACKET_DATA_INFO HTTransmitMode HTBurstCount HTDataLength HTFillPattern HTRun
L3mod_stream_array.tcl	Creates a source IP stream on a Layer 3 card, adds additional streams, and modifies the packet length field.	HTSetStructure with L3_DEFINE_IP_STREAM L3_DEFINE_MULTI_IP_STREAM L3_MOD_STREAMS_ARRAY
L3PingCount.tcl	Sets up the Layer 3 stack on an ML-7710 SmartCard.	HTSetSpeed HTDuplexMode HTTransmitMode HTBurstcount HTClearPort HTLayer3SetAddress HTRun HTGetEnhancedCounters
L3stack.tcl	Configures the SmartCard's local MAC address, IP address, gateway IP address, and PING target address.	HTLayer3SetAddress
L3Trigger.tcl	Illustrates the use of triggers on Layer 3 cards.	HTResetPort HGSetGroup HGAddtoGroup HGSetSpeed HTTransmitMode HTBurstCount HTTrigger HTGetStructure with L3_CAPTURE_PACKET_DATA_INFO L3_CAPTURE_COUNT_INFO HTSetStructure with L3_DEFINE_IP_STREAM L3_DEFINE_MULTI_IP_STREAM HTSetCommand with L3_CAPTURE_OFF_TYPE L3_CAPTURE_ALL_TYPE L3_CAPTURE_TRIGGERS_TYPE

L3Zero.tcl	Shows the number of streams on the target L3 card and prompts the user if the streams should be removed. Useful when checking other programs to validate the number of created L3 streams.	HTGetStructure <i>with</i> L3_DEFINE_STREAM_COUNT_INFO HTSetStructure <i>with</i> L3_DEFINE_SMARTBITS_STREAM
SBSStream.tcl	Script called by an external program to create a SmartBits type stream, with iHub iSlot iPort and DATA_LENGTH values set by the calling program.	HTSetStructure <i>with</i> L3_DEFINE_SMARTBITS_STREAM
udpstream.tcl	Creates multiple UDP streams.	HTSetStructure <i>with</i> L3_DEFINE_UDP_STREAM L3_DEFINE_MULTI_UDP_STREAM

LibX

The LibX Utilities are a family of Tcl procedures, created with the Programming Library, that simplify setting up and operating cards and chassis. Developed for technical support, they provide a quick, command line-based method to set up cards, display essential configuration parameters, and run test programs.

See *Using LibX for Simplified Library Control* (page 85) for complete information, as well as to the *ReadMe.pdf* file in the LibX directory.

The LibX directory contains the following files:

File Name	Description	Demonstrates
l3x.tcl	Layer 3 extension procedures for the programming library.	See <i>Using LibX for Simplified Library Control</i> (page 85) for information.
libx.tcl	Basic extension procedures for the programming library.	
loadx.tcl	Loads the l3x.tcl and libx.tcl extensions.	
unloadx.tcl	Unloads the l3x.tcl and libx.tcl extensions.	

Packet Over Sonet (POS)

This script illustrates how to work with the Packet Over Sonet (POS) SmartModules (POS-6500A/POS-3500A and POS-6500B/POS-3500B).

File Name	Description	Demonstrates
L3Extension.tcl	Simple program using two POS-3500A SmartModules connected back-to-back in a SmartBits 600.	HGSetGroup HGAddtoGroup HGIsHubSlotPortInGroup HGClearPort HGStart HGGetCounters HTSetStructure <i>with</i> POS_CARD_PORT_ENCAP L3_DEFINE_IP_STREAM L3_DEFINE_IP_STREAM_EXTENSION

SmartAPI for SmartApplications (SmartAPI)

This script works with the SmartLib SmartAPI for SmartApplications.

File Name	Description	Demonstrates
SmartAPI.tcl	Demonstrates the four SmartApplications tests: Throughput, Back-to-Back, Packet Loss, and Latency.	<i>Refer to the SmartAPI for SmartApplications User Guide for information on these tests.</i>

SmartBits 6000 (SMB6000)

This script works with SmartBits 6000/600 and 2000/200 systems.

File Name	Description	Demonstrates
Controller.tcl	Illustrates differences between the SMB2000/200 and SMB6000/600 chassis types, along with the two port-mapping modes: Compatible and Native.	ETSocketLink ETGetProduct Family ETGetController NSSetPortMappingMode ETUnLink

Token Ring (TokenRing)

This script works with Token Ring SmartCards.

File Name	Description	Demonstrates
TokenRing.tcl	Sets up transmission parameters and two VFDs on a Token Ring SmartCard	HTResetPort HTSetTokenRingProperty HTVFD

Examples in the smartlib_tcl Directory

These examples make use of the new, improved Tcl interface supported by the smartlib.tcl interface file. They mirror the functionality of the samples in /Samples/et1000_tcl/ with the exception that they use the default values file. For this reason, some configuration settings may be different.

For detailed information about these examples, see the corresponding /et1000_tcl/ entry in the tables above. The new Tcl interface files under /Samples/smartlib_tcl/ correspond to the samples contained in the following et1000_tcl directories:

- /GIG
- /POS
- /Layer3
- /SMB6000

Examples of Functions in the Tcl Code Examples

The table below lists instances of Original Functions and Message Functions in the sample code. For examples for GIG, Layer3, POS, and SMB 6000 using the new Tcl interface, see the /Samples/smartlib_tcl directory.

Function Name	Sample Directory	Code Sample Name
ETBurst	et1000	ETVFD_CYCLE.TCL
ETCaptureParams	et1000	multi.tcl
ETDataLength	et1000	ET1000MODE.tcl
ETDataPattern	et1000	ET1000MODE.tcl
ETGetCapturePacket	et1000	multi.tcl
ETGetCounters	et1000	multi.tcl
ETGetLibVersion	AllCards	LibVer.tcl
	AllCards	Startup.tcl
ETLink / ETUnLink	AllCards	1stlink.tcl
	AllCards	multi-link.tcl
ETMake2DArray	AllCards	cardmod.tcl
ETReceiveTrigger	et1000	multi.tcl
ETMFCounter	et1000	multi.tcl
ETRun	et1000	ET1000MODE.tcl
ETSetCurrentLink	AllCards	multi-link.tcl
ETSetSel	et1000	ET1000MODE.tcl
ETSocketLink / ETUnLink	AllCards	SocketLink.tcl
ETVFDParams	et1000	ET1000MODE.tcl
ETVFDRun	et1000	ET1000MODE.tcl
HGAddtoGroup	AllCards	Backoff.tcl
HGAddtoGroup	AllCards	Group.tcl
	AllCards	GroupCount.tcl
	AllCards	Trigger.tcl
	ATMCard	ATMGetCounts.tcl
	FastCard	Collision.tcl
	FastCard	FastTrig&VFD.tcl
	ETH	setspeed.tcl
	GIG	GIGCount.tcl
	Layer3	L3Trigger.tcl
HGClearGroup	AllCards	Backoff.tcl
	AllCards	Group.tcl
	ATMCard	ATMGetCounts.tcl
	ETH	setspeed.tcl
HGClearPort	FastCard	Collision.tcl
	GIG	GIGCount.tcl
HGCollisionBackoffAggressiveness	AllCards	Backoff.tcl
HGDataLength	ETH	setspeed.tcl
HGDuplexMode	FastCard	Collision.tcl
	ETH	setspeed.tcl
HGGetCounters	AllCards	GroupCount.tcl
	AllCards	Trigger.tcl

Function Name	Sample Directory	Code Sample Name
	FastCard	Collision.tcl
	GIG	GIGCount.tcl
HGIsHubSlotPortInGroup	AllCards	Trigger.tcl
HGResetPort	AllCards	Backoff.tcl
	AllCards	Group.tcl
	ETH	setspeed.tcl
HGSetGroup	AllCards	GroupCount.tcl
	AllCards	Trigger.tcl
	ATMCard	ATMGetCounts.tcl
	FastCard	Collision.tcl
	FastCard	FastTrig&VFD.tcl
	GIG	GIGCount.tcl
	Layer3	L3Trigger.tcl
HGSetSpeed	ETH	setspeed.tcl
	Layer3	setspeed.tcl
	Layer3	L3Trigger.tcl
HGStart / HGStop	AllCards	Group.tcl
	AllCards	Backoff.tcl
	AllCards	GroupCount.tcl
	ATMCard	ATMGetCounts.tcl
	ETH	setspeed.tcl
	GIG	GIGCount.tcl
HGTransmitMode	FastCard	capture.tcl
	FastCard	Collision.tcl
	FastCard	FastTrig&VFD.tcl
	FastCard	gap.tcl
	FastCard	setspeed.tcl
	Layer3	L3_V2_HIST_LAT.tcl
	Layer3	l3min.tcl
	Layer3	L3PingCount.tcl
	Layer3	L3Trigger.tcl
	ETH	setspeed.tcl
HGVFD	AllCards	Trigger.tcl
HTBurstCount	AllCards	capture.tcl
	FastCard	gap.tcl
	FastCard	Collision.tcl
	FastCard	FastTrig&VFD.tcl
	Layer3	L3_V2_HIST_LAT.tcl
	Layer3	l3min.tcl
	Layer3	L3PingCount.tcl
	Layer3	L3Trigger.tcl
HTCardModels	AllCards	cardmod.tcl
HTClearPort	AllCards	GapPerCent.tcl
	Layer3	L3PingCount.tcl
HTCollision	FastCard	Collision.tcl
HTCountStructure	AllCards	gap.tcl

Function Name	Sample Directory	Code Sample Name
HTDataLength	AllCards	GapPerCent.tcl
	AllCards	gap.tcl
	FastCard	capture.tcl
	AllCards	GapPerCent.tcl
	FastCard	gap.tcl
	ETH	setspeed.tcl
HTDuplexMode	AllCards	gap.tcl
	FastCard	gap.tcl
	ETH	setspeed.tcl
	Layer3	L3PingCount.tcl
HTFillPattern	AllCards	vfd.tcl
	FastCard	capture.tcl
	FastCard	FastTrig&VFD.tcl
	Layer3	l3min.tcl
HTFindMIIAddress	ETH	mii.tcl
HTGap	AllCards	gap.tcl
	AllCards	GapPerCent.tcl
	FastCard	gap.tcl
HTGetCardModel	FastCard	EnhancedStats.tcl
HTGetCounters	AllCards	gap.tcl
	AllCards	GapPerCent.tcl
	FastCard	FastTrig&VFD.tcl
	FastCard	gap.tcl
	FastCard	gap.tcl
	FastCard	gap.tcl
HTGetEnhancedCounters	Layer3	L3PingCount.tcl
HTGetEnhancedStatus	ATM	ATM_Status.tcl
	FastCard	EnhancedStats.tcl
HTGetLED	ATMCard	ATM_Status.tcl
HTGetStructure <i>with:</i>		
ATM_CARD_CAPABILITY	ATMCard	ATMGetCounts.tcl
ATM_CARD_CAPABILITY	ATMCard	ATM_Clip.tcl
ATM_CARD_CAPABILITY	ATMCard	ATM_SET&Count.tcl
	ATMCard	atm_trig.tcl
	ATMCard	ATMData.tcl
ATM_CARD_INFO	ATMCard	ATMData.tcl
ATM_CARD_TYPE	ATMCard	ATMData.tcl
ATM_CONN_TRIGGER_INFO	ATMCard	atm_trig.tcl
ATM_SAAL_INFO	ATMCard	ATM_SAAL.tcl
ATM_SONET_INFO	ATMCard	ATMSonetInfo.tcl
ATM_STREAM_DETAIL	ATMCard	ATM_SET&Count.tcl
ATM_STREAM_DETAIL_INFO	ATMCard	ATM_SET&Count.tcl
ATM_VCC_INFO	ATMCard	ATM_SET&Count.tcl
	ATMCard	atm_trig.tcl
ETH_CARD_INFO	ETH	ETHCardInfo.tcl

Function Name	Sample Directory	Code Sample Name
L3_CAPTURE_COUNT_INFO	Layer3	l3min.tcl
	Layer3	L3Trigger.tcl
L3_CAPTURE_PACKET_DATA_INFO	Layer3	l3min.tcl
HTLayer3SetAddress	Layer3	L3PingCount.tcl
HTReadMII	ETH	mii.tcl
HTResetPort	AllCards	gap.tcl
	FastCard	FastTrig&VFD.tcl
	FastCard	gap.tcl
	ETH	mii.tcl
	ETH	setspeed.tcl
HTRun	Layer3	L3Trigger.tcl
	AllCards	gap.tcl
	AllCards	GapPerCent.tcl
	AllCards	multi-link.tcl
	FastCard	Collision.tcl
	FastCard	gap.tcl
	ETH	setspeed.tcl
	GIG	GIGVFD.tcl
	Layer3	L3_V2_HIST_LAT.tcl
	Layer3	l3min.tcl
HTSelectReceive	et1000	ET1000MODE.tcl
	et1000	ET1000MODE.tcl
HTSelectTransmit	et1000	ET1000MODE.tcl
HTSetCommand <i>with:</i>		
ATM_ILMI_REGISTER	ATM	ATM_ILMI_REGISTER
ATM_ILMI_DEREGISTER	ATM	ATM_ILMI_REGISTER
ATM_SAAL_ESTABLISH	ATM	ATM_SAAL.tcl
L3_HIST_V2_LATENCY	Layer3	L3_V2_HIST_LAT.tcl
L3_START_ARPS	Layer3	l3min.tcl
L3_CAPTURE_OFF_TYPE	Layer3	l3min.tcl
	Layer3	L3Trigger.tcl
L3_CAPTURE_ALL_TYPE	Layer3	l3min.tcl
	Layer3	L3Trigger.tcl
L3_CAPTURE_TRIGGERS_TYPE	Layer3	L3Trigger.tcl

HTSetSpeed	AllCards	GapPerCent.tcl
	FastCard	gap.tcl
	ETH	setspeed.tcl
	Layer3	L3PingCount.tcl
HTSetStructure <i>with:</i>		
ATM_STREAM_CONTROL	ATMCard	ATMGetCounts.tcl
ATM_STREAM_CONTROL	ATMCard	ATM_Clip.tcl
ATM_STREAM_CONTROL	ATMCard	atm_trig.tcl
	ATMCard	ATM_SET&Count.tcl
ATM_STREAM	ATMCard	ATM_Clip.tcl
ATM_STREAM	ATMCard	ATM_SET&Count.tcl
	ATMCard	atm_trig.tcl
ATM_FRAME_DEF	ATMCard	ATM_SET&Count.tcl
ATM_FRAME_DEF	ATMCard	ATM_Clip.tcl
ATM_FRAME_DEF	ATMCard	ATMGetCounts.tcl
ETH_TRANSMIT	ETH	ETHTransmit.tcl
FST_ALTERNATE_TX	FastCard	capture.tcl
FST_CAPTURE_COUNT_INFO	FastCard	capture.tcl
FST_CAPTURE_PARAMS	FastCard	capture.tcl
GIG_STRUC_TX	GIG	GIGCount.tcl
GIG_STRUC_FILL_PATTERN	GIG	GIGVFD.tcl
GIG_STRUC_VFD3	GIG	GIGVFD.tcl
GIG_STRUC_CAPTURE_SETUP	GIG	GIGVFD.tcl
GIG_STRUC_CAP_DATA_INFO	GIG	GIGVFD.tcl
L3_DEFINE_IP_STREAM	Layer3	ipstream.tcl
	Layer3	L3Trigger.tcl
	Layer3	L3mod_stream_array.tcl
L3_DEFINE_MULTI_IP_STREAM	Layer3	ipstream.tcl
	Layer3	L3mod_stream_array.tcl
	Layer3	L3Trigger.tcl
L3_DEFINE_IPX_STREAM	Layer3	ipxstream.tcl
L3_DEFINE_MULTI_IPX_STREAM	Layer3	ipxstream.tcl
L3_DEFINE_SMARTBITS_STREAM	Layer3	SBSStream.tcl
L3_DEFINE_UDP_STREAM	Layer3	udpstream.tcl
L3_DEFINE_MULTI_UDP_STREAM	Layer3	udpstream.tcl
L3_MOD_STREAMS_ARRAY	Layer3	L3mod_stream_array.tcl
HTTrigger	AllCards	Trigger.tcl
	FastCard	FastTrig&VFD.tcl
	Layer3	L3Trigger.tcl
HTVFD	AllCards	vfd.tcl
HTWriteMII	ETH	mii.tcl

C Demo Modules

The Samples\C subdirectory includes demo modules for ATM, the LAN-6100A SmartModule, Layer 2, and Layer 3.

These modules are divided into steps to enable you to see the actions needed for basic testing with SmartLib.

ATM (Atm)

This module demonstrates PPP over ATM for ATM-2 SmartCards (AT-9155c and AT-9622).

LAN-6100A (Lan6100a)

This module demonstrates how to work with the LAN-6100A SmartModule. The demo is based on the following steps.

- Step 1** **Connect to SmartBits** shows how to use `NSSocketLink()` to connect to the SmartBits chassis, using the Native port mapping mode. (Throughout the demo, the equivalent message functions for the Compatible port mapping are also shown as comments.)
- Step 2** **Learn Target Port MAC Addresses** shows how to generate ARP and PING frames and to **retrieve** the MAC address of the device with the target IP address.
- Step 3** **Select Transmit and Receive Ports** shows a simple algorithm that is used to pair up the ports and verify the results of the management frames generated in Step 2, based on the table `PortAddrTable` `PreSettingPortsTable [NumPorts]` defined in the `demo.h` file.
- Step 4** **Set up Burst of IP Frames** demonstrates how to use `NSCreateFrame()`, `NSModifyFrame()`, and `HTFrame()` to configure IP traffic on the transmit ports.
- Step 5** **Test Layer 2 and Layer 3 Counters** demonstrates the counters on the LAN-6100A. using `HTGetStructure` with the `ETH_COUNTER_INFO` and `FST_PROTOCOL_COUNTER_INFO` `iTypes`.
- Step 6** **Test IP Checksum** demonstrates the ability of SmartLib to generate IP traffic with invalid IP Checksums and the LAN-6100A's ability to count these packets, as well as to calculate the correct checksum when configured to do so.
- Step 7** **Display Capture Data** shows the **capture** features of the LAN-6100A and backward compatibility with the SX-7410.
- Step 8** **Disconnect and Clean-up** is self-explanatory.

Layer 2

The Layer 2 module demonstrates how to set up unidirectional traffic between source and destination with cards in Traditional (Layer 2) mode. Card configurations cover each type of card.

Traditional mode uses VFDs (Variable Field Definitions) to set up one or more traffic streams. ARP responses and histogram results are not available. Complex Layer 2, 3, and 4 testing is more difficult than in SmartMetrics mode.

Layer 3

The Layer 3 module demonstrates how to set up unidirectional traffic between source and destination with cards in SmartMetrics mode. Card configurations cover each type of card that supports SmartMetrics. In SmartMetrics mode, a card uses VFDs as well as the more complex VTEs to set up traffic streams. ARP responses and other network interactions are automatic. Relational histogram results are available.

A card running in SmartMetrics mode provides a number of important capabilities, including:

- Multiple streams configurable on a per-stream basis
- True ARP interaction
- Histogram results

A different histogram is enabled on the card, depending on whether you wish to view Latency over Time, Latency per Stream, Sequence Tracking, and so on.

Common Steps in the Layer 2 and Layer 3 C Demos

The Layer 2 and Layer 3 C demos share five basic steps, as shown in the Tcl snippet below. (This is taken from the main routine in demo.cpp on the installation CD.)

```
Step1_ExamineSystem();           /* Show card types, models and version
                                information */
Step2_DetermineConnections();    /* Determine connections */
Step3_ResetAndSetupAll();       /* Reset and setup each card */

Step4_Transmit(STAGGERED_START); /* Transmit packets */
Step5_ShowAllCounters();        /* Show counters */

Step4_Transmit(SYNCHRONIZED_START); /* Transmit packets */
Step5_ShowAllCounters();        /* Show counters */

/* Terminate our session with SmartBits */
printf("\nPress any key to UnLink and close the window\n");
```

The five steps are the following:

- Step 1** Query the cards to determine what the kinds of cards are present in the chassis.
- Step 2** Order cards by pairs, so that each destination card has a source card.
- ❖ For Layer2 and Layer3 demos, the next card of the same type is configured as the destination card. Unpaired cards are not used in the test.
 - ❖ Cards in the SmartAPI demo are paired according to the test configuration.
- Step 3** Set cards to a known state, then sends the proper configuration for each card type. Step 3 is the most complex step of this demo series.
- Step 4** Start the test traffic.
- Step 5** Display test results.

Files for C Demos

Each C demo contains the following file types. Each module uses a slightly different set of files; all modules, however, have common features.

NOTE For the Layer 3 Demo, there is no `cpp` file. You display results using one of the histogram modules.

File	Description
Demo*.cpp (or) *Main.cpp	The central file used to link the PC to the SmartBits chassis and call the test Steps.
Step1.cpp (through) Step5.cpp	Files containing code that executes the demo steps. See <i>Common Steps in the C Demos</i> above for definitions.
Utils.cpp	A catch-all file that contains, for example, certain constants, error code, and the routines used for reading from the *.ini files found in Smart Signaling and SmartAPI.
*.h	The header file for the individual demo project.

Running the C Demos

Use the following steps to compile and run the C demos.

MS Windows

1. Create a 32-bit Console Applications Project.
2. Add the source files for the desired demo and the appropriate Netcom Systems library into the project
3. Compile the demo.
4. Run the program.

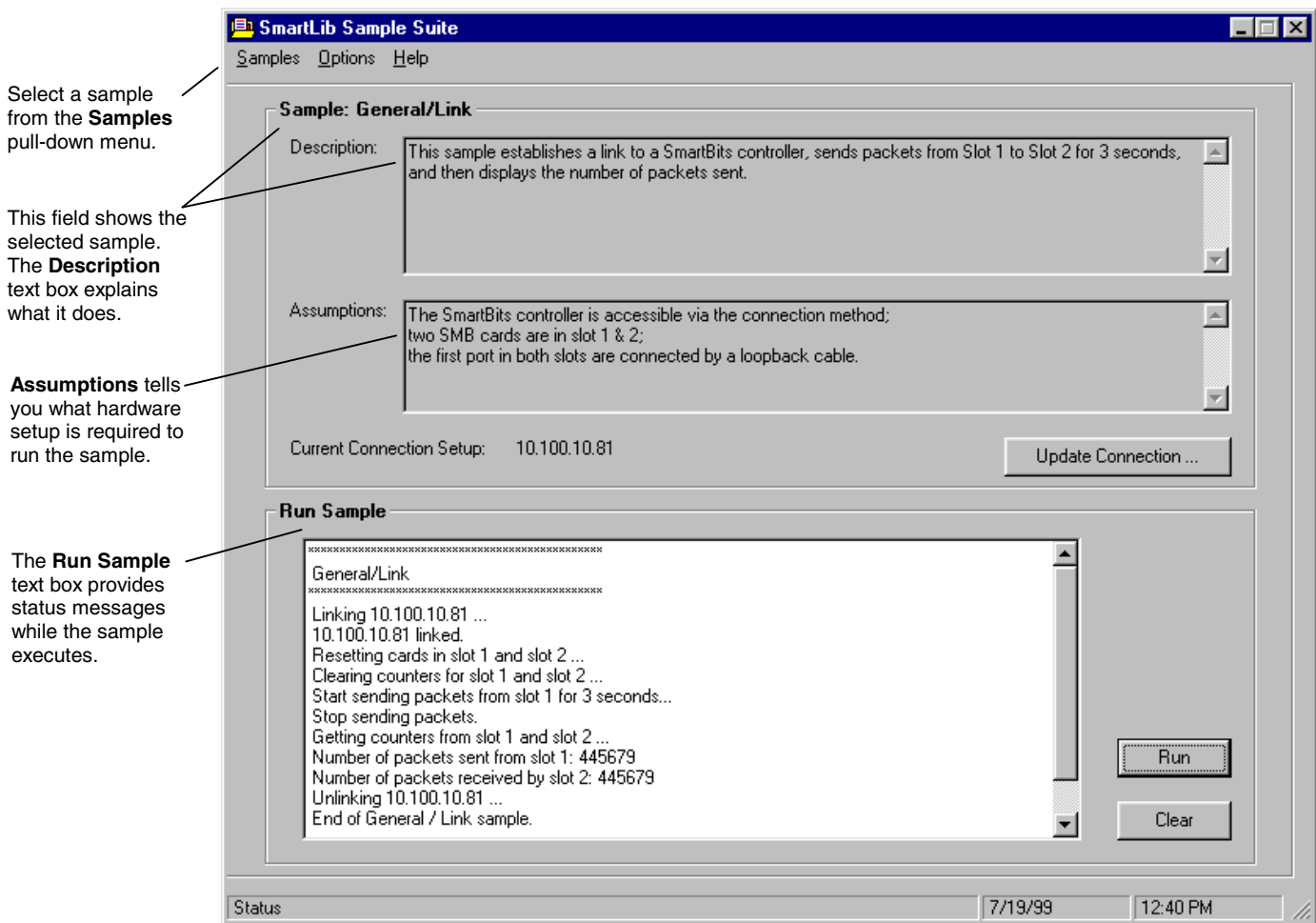
UNIX

1. Create a makefile that compiles all `cpp` files, then links them with `libetsmb.so` to produce an executable.
2. Run the executable.

Visual Basic (VB) Demo

The Visual Basic demo module is intended to show basic test procedures when using Microsoft Visual Basic.

To launch the demo, go to the `SmartLib\Samples\VB` directory from the SmartLib CD and double-click on the `sample.exe` icon. The **SmartLib Sample Suite** window opens. (To view the source code, launch `sample.vbp`.)



How to Run the Demo

Use the following steps to run a VB sample.

1. From the **Samples** pull-down menu, select a sample to run. Currently, one sample is supported: under **General**, a module named **Link**. This is a basic task that links a PC to a SmartBits chassis. (Other procedures will be added in subsequent SmartLib releases.)

NOTE **General/Link** is the default sample when you first launch the sample shell. To run the sample, you need only make sure that the connection type is correct, and then click **Run**.

2. The **Description** text box explains the sample, which is identified by the heading (here, **General/Link**).
3. The **Assumptions** text box tells you what hardware setup is needed to run the procedure.
4. **Current Connection Setup** shows the current interface between the PC and the SmartBits. Here, an IP socket connection is being used, and the IP address defined for SmartBits is **10.100.10.81**.
5. To update the connection type, select **Update Connection**. Then use the window options to specify either Serial port connection or Ethernet connection with the SmartBits chassis IP address.
6. Select **Run** to start the sample.
7. The **Run Sample** text box displays status messages as the procedure runs.

Setting Break Points

You can view the actual SmartLib functions that are executed by the sample by going to the VB debug mode and setting a break point at the **Run** command. Doing this can show you in detail how Library functions are used in each sample.

Chapter 11:

Using LibX for Simplified Library Control

The LibX Utilities are a family of Tcl procedures, created with the Netcom Systems Programming Library and designed to simplify the process of setting up and operating Netcom Systems cards and chassis.

These procedures were developed for Technical Support to allow quick, command line-based procedures to set up SmartCards and display essential card configuration parameters and test-program output.

This chapter briefly explains fundamental concepts for LibX, and also includes simple exercises that illustrate each concept. These examples are identified by the **Try It!** heading.

Commands you enter are shown in bold **courier type**—for example, **set_capture \$card1**.

All the examples in this chapter were run on a SmartBits 200 with four ML-7710 SmartMetrics SmartCards connected to a Windows 95 PC.

System Requirements

LibX requires Tcl version 8.0 or higher, because it makes use of capabilities such as namespaces that were introduced with Tcl 8.0. LibX was introduced as part of the Programming Library sample code with the 3.07 release and has been formally tested only against that release.

Cards Supported

The current release of LibX focuses primarily on Ethernet cards in SmartMetrics mode. However, many of the functions described here work across a range of card types.

LibX Components

LibX consists of five files:

- loadx.tcl A Tcl script that loads the LibX components.
- libx.tcl Core LibX commands and definitions.
- l3x.tcl Layer 3 / SmartMetrics stream commands.
- unloadx.tcl A Tcl script that unloads the LibX components.
- readme.doc An online version of the information in this chapter.

Example 1 – Loading LibX with loadx.tcl

Before installing LibX you must start the Tcl Shell and source `et1000.tcl` to install the Netcom Systems programming library.

Try It! – Load LibX

- Start the Tcl Shell and source `et1000.tcl`.
- `cd` to the directory holding the LibX files.
- Enter `source loadx.tcl`. If the command succeeds the message `LibX loaded` will be displayed as shown below.



```
tclsh80
Auto
% source loadx.tcl
LibX loaded
%
```

How Does It Work?

After a series of checks, `loadx.tcl` sources the component files `libx.tcl` and `l3x.tcl`, then imports only the procedure names from each into the global namespace.

Load Problems

If the message was not displayed, there was a problem installing LibX. Possible error messages include:

```
ERROR libx requires Tcl version 8.0 or higher
```

You are using a version of Tcl older than 8.0. You must upgrade before using LibX.

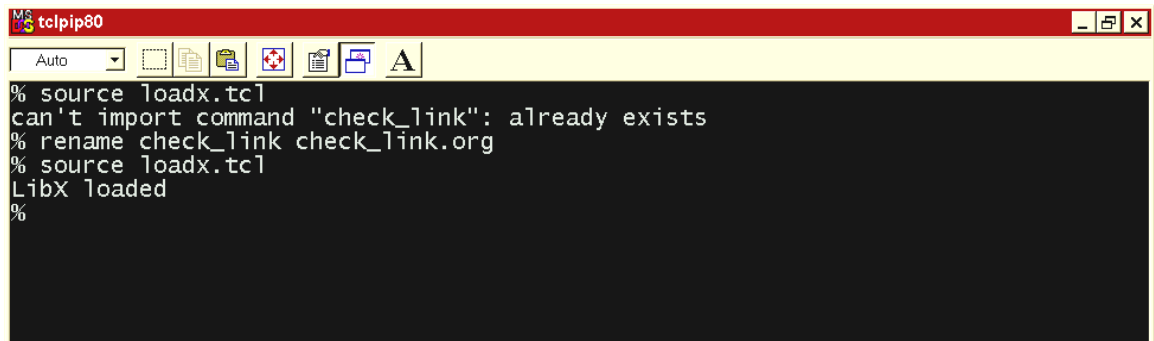
```
Files libx.tcl and/or l3x.tcl were not found
```

The component files are not in the local directory. Check the current directory (using `pwd`) and be sure it is the directory that contains the LibX files.

```
Can't import command "check_link": already exists
```

There is a command in the global namespace that has the same name as one of the LibX commands. You have to remove or rename the existing command before you can load LibX. This is a safety feature of LibX that prevents it from silently replacing a command or procedure that already exists.

To remove a conflict of this type, use the Tcl `rename` command. For example, to rename the procedure `check_link` as `check_link.org`, use `rename check_link check_link.org`. The sequence is shown below.

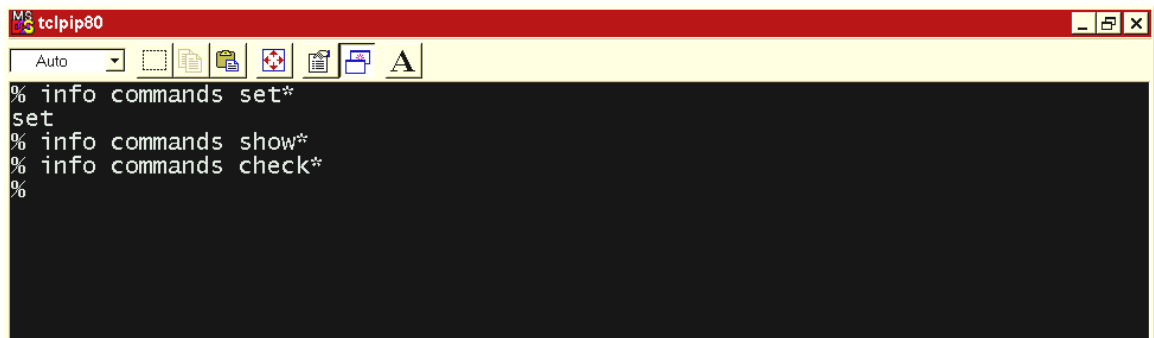


```
MS tcip80
Auto
% source loadx.tcl
can't import command "check_link": already exists
% rename check_link check_link.org
% source loadx.tcl
LibX loaded
%
%
```

Listing Commands

LibX adds a series of new commands to the Tcl environment.

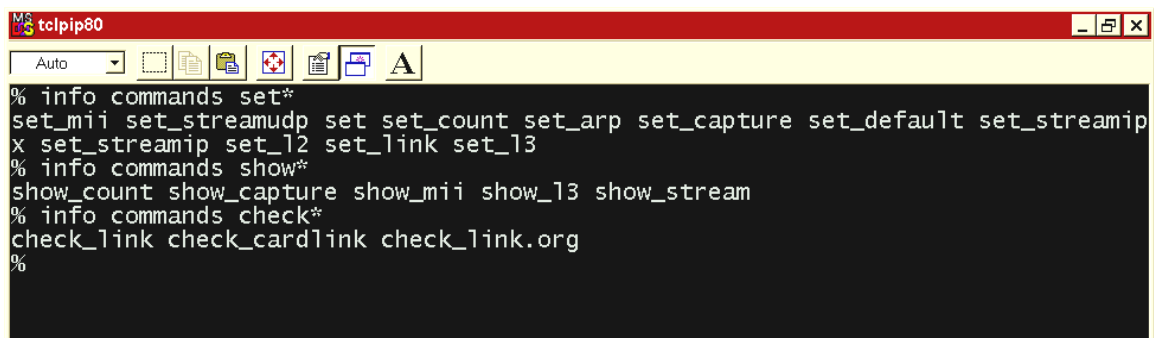
Before starting, you could check for all the commands that start with **set**, **show**, or **check** as follows:



```
MS tcip80
Auto
% info commands set*
set
% info commands show*
% info commands check*
%
%
```

The only command above is the Tcl **set** command.

After installing LibX, if we repeat the same command sequence, we get:



```
MS tcip80
Auto
% info commands set*
set_mii set_streamudp set set_count set_arp set_capture set_default set_streamip
x set_streamip set_12 set_link set_13
% info commands show*
show_count show_capture show_mii show_13 show_stream
% info commands check*
check_link check_cardlink check_link.org
%
%
```

These new commands are LibX.

Command Format

All LibX commands have the same format:

command root => one of: set_ (or) show_ (or) check_

The command root is appended to the appropriate action, such as:

set_streamip	Sets up IP data streams on the target card.s
set_link	Connects to the SmartBits chassis.
set_capture	Sets up and starts a capture on the target card.
show_stream	Shows the streams currently established on the target card.

For simplicity and consistency, all LibX commands have the underscore character (`_`) only following the command root and nowhere else. Commands are always lower-case and always singular, never plural.

See the summary table at the end of this chapter for a complete list of the currently defined LibX commands.

Card Numbering Conventions

Some LibX commands, such as **set_link**, have no arguments. Most commands, however, have at least the ID of target SmartCard.

The Programming Library identifies a card's position in the chassis or chassis stack by the Hub Slot Port triple. This numbering system is zero-based, so the first card in the first chassis is identified as Hub Slot Port 0 0 0.

LibX simplifies this by defining each potential chassis position with a card number. For example, the first card in the first chassis (Hub Slot Port 0 0 0) is defined as `card1`.

You can see how `card1` is defined using the **puts** command.



```
tclpip80
Auto
% puts $card1
{0 0 0}
% _
```

Note that the `{0 0 0}` is *not* the same as `0 0 0`. The LibX commands expect a list to identify a card, not individual numbers. Using lists to reference cards rather than the Hub Slot Port triple allows us to expand the concept of groups.

Example To set capture on the first card in the first chassis, you would enter:

```
set_capture $card1
```

Since the LibX commands accept a list as an argument, you can create a group as a list of lists and pass the group list to the LibX command exactly as you would a single card.

Currently LibX does not include a **set_group** command; you have to create your own. The usage is conceptually the same as to define a single card. For example, to set up a group called **txgroup** with cards 1, 3, 4, and 12 as members, then start a capture on all group members, you would use:

```
set txgroup [list $card1 $card3 $card4 $card12]
set_capture $txgroup
```

This will set capture on card 1, 3, 4, and 12. (Although the library has group commands for many functions, capture is not one of them.)

In cases where the parent library does not have a group capability (such as setting a capture), LibX will create an appropriate loop mechanism, usually a Tcl **foreach** command, to achieve this purpose. In cases where the parent library already provides a group command, LibX will convert the card list into a SmartLib group, then execute the appropriate group command.

Example 2 – set_default and set_link

LibX has a **set_default** procedure that sets the card to a predefined base state. The base state is approximately the base state established by the SmartWindow application when you select **File>New**.

This is different from the default state established on power-up or by using the library `HTRResetPort $RESET_FULLL` command. For example, a power-up or `ResetPort` will set the background pattern for an Ethernet card to all zero. SmartWindow and **set_default** create a basic Ethernet frame. For example, for card 2 it will set the destination MAC address to FF FF FF FF FF FF and the source MAC address to 00 00 00 00 00 02.

With the power-on default of all zero, you generally will *not* be able to send packets through a switch, since almost any switch will block an all-zero packet. With the SmartWindow or **set_default** packet, in contrast, the packets will pass.

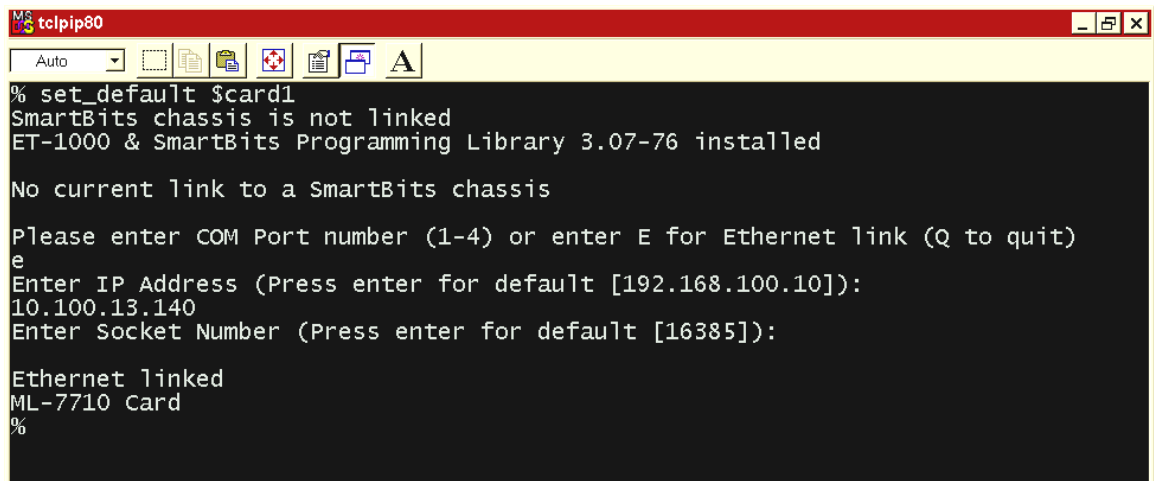
Try It! – set_default

- Set up two SmartCards in slots 1 and 2.
- Power on the chassis.
- Start the Tcl Shell and source `et1000.tcl`.
- Set card 1 to the default state with `set_default $card1`.

Normally, you would not be able to send a command to a SmartBits chassis, since you didn't link. Attempting to send a command before linking would produce an error.

LibX always checks the system's state before executing. If you are not linked, it will prompt for a serial port or Ethernet address and link before executing the command, as shown below.

- When prompted, enter the COM port number to connect via serial connection, or enter **E** to select an Ethernet link. If linking via Ethernet, enter the IP address and TCP port number. After the link is complete (as indicated by the `Ethernet linked` message), the card type is displayed by the **set_default** procedure.



```
MS tclpip80
Auto
% set_default $card1
SmartBits chassis is not linked
ET-1000 & SmartBits Programming Library 3.07-76 installed

No current link to a SmartBits chassis

Please enter COM Port number (1-4) or enter E for Ethernet link (Q to quit)
e
Enter IP Address (Press enter for default [192.168.100.10]):
10.100.13.140
Enter Socket Number (Press enter for default [16385]):

Ethernet linked
ML-7710 Card
%
```

How Does It Work?

If you look at the **set_default** procedure (in `libx.tcl`), you will see that it starts as follows:

```
proc set_default { {group} } {
    check_link
```

The **set_default** procedure has one argument, **group**, which we have seen is the list defining the card—in this case, **\$card1**. The first step is to call the procedure **check_link**, which checks the link state. If it is not linked, it calls a second procedure, **set_link**, which actually sets up the link.

You may want to change the default Ethernet address in **set_link**. The easiest way is to open `libx.tcl` in a programming editor and search for the line:

```
set IPADDRESS "192.168.100.10"
```

Change it to the desired address. Reload LibX and the new default will be the value you entered.

The **set_default** procedure identifies the target card using `HTGetCardModel`. Based upon the name returned it runs the default setting procedure appropriate to the card. A part of the **set_default** procedure is shown below:

```
set cardname ""
LIBCMD HTGetCardModel cardname $H $S $P
switch $cardname {
SE-6205 -
SC-6305 -
ST-6405 -
ST-6410 {puts "L2 10Mb Ethernet"
set_ethernet $H $S $P}
SX-7205 -
SX-7210 -
```

Example 3 – Transmit and Count

Three commands control transmission in LibX:

- **run** Starts transmission on the target card.
- **stop** Stops transmission on the target card.
- **burst** Sends a burst from the target card.

Transmit

Try It! – Run and Stop

- Ensure the Tcl Shell is started and the library is loaded.
- Connect card 1 and card 2 either back to back with a crossover cable or through a device.
- Start card 1 transmitting with `run $card1`. The TX LED will light on card 1 and the RX LED will light on card 2.
- Stop card 1 with `stop $card1`. The LEDs will go out.

How Does It Work?

The transmit procedures are some of the simplest in LibX. Here is the **run** procedure in its entirety:

```
proc run {group} {
    check_link
    list2group $group
    LIBCMD HGStart
}
```

Here is what the procedure does:

Command / Argument	Description
run	Procedure name.
group	The only argument: a card or a group of cards.
check_link	Checks for the link (as described above).
list2group \$group	Calls an internal procedure that converts the group list into a SmartBits group.
HGStart	This group command starts the group transmitting.

Counters

Two LibX procedures give control over the card counters:

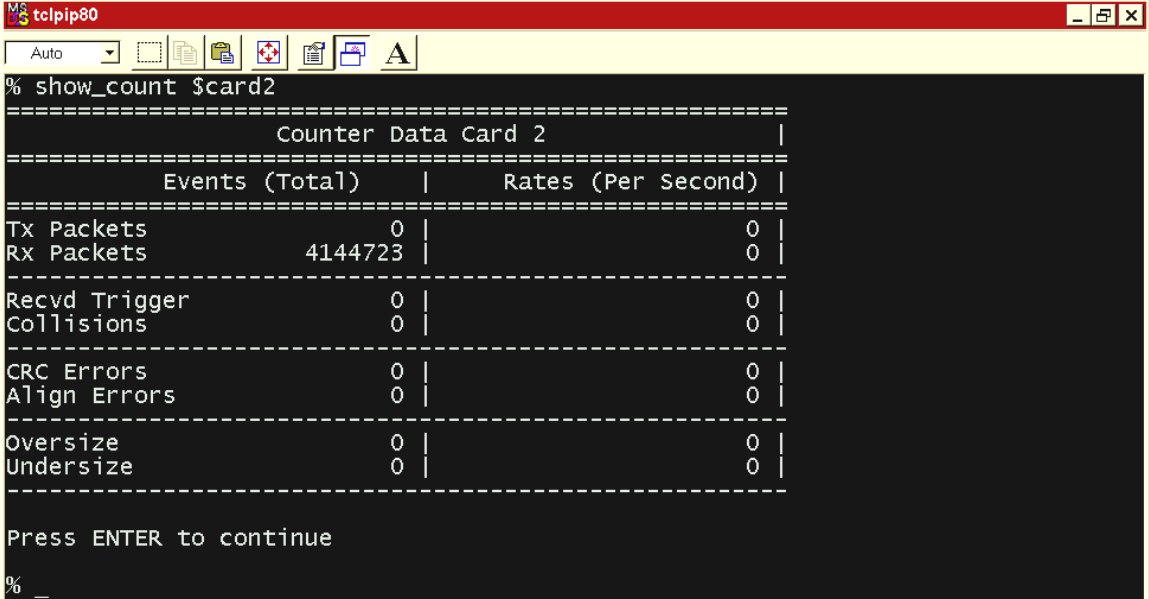
- **set_count** Clears the on card counters.
- **show_count** Displays the current card counter data.

NOTE SmartBits counters are always enabled. At any time, you can retrieve the counter data, and the data will show all the counts since the counters were last reset or since the chassis was powered on.

Try It! – Counters

After ensuring the system is ready to send commands, do the following:

- Read the counter data on card 2 by entering `show_count $card2`. The counter data from card 2 will be displayed as shown. Here, card 2 has received a little over four million packets but hasn't transmitted anything.
- Clear the counters on card 2 by entering `set_count $card2`. Now use `show_count` to display the counter data again showing all the counts have been reset to zero.
- Start card 1 transmitting using the `run` command. While the card is still transmitting, display the counts on card 2 with `show_count`.
- The `show_count` display will now show the packets received (Events) plus the packets-per-second rate (Rates).



```
MS tcPIP80
Auto
% show_count $card2
=====
Counter Data Card 2
=====
Events (Total) | Rates (Per Second) |
=====
Tx Packets      0 | 0 |
RX Packets     4144723 | 0 |
-----
Recvd Trigger   0 | 0 |
Collisions      0 | 0 |
-----
CRC Errors      0 | 0 |
Align Errors    0 | 0 |
-----
Oversize        0 | 0 |
Undersize       0 | 0 |
-----
Press ENTER to continue
% _
```

- Leave card 1 transmitting. Show the counter data from card 1 this time, using `show_count`. How is the counter data from card 1 different?

Answer: Card 1 will show transmitted packets and no received packets. Card 2 shows received packets and no transmitted packets.

- While card 1 is still transmitting, start card2 transmitting using the **run** command. Display the counter data for both cards. If the cards are in half-duplex mode, you will now see the collision and undersize counts increasing as shown. You may or may not see a trigger count depending on what state the background on the card is in.

```

MS tcpip80
Auto
Press ENTER to continue
% show_count $card2
=====
Counter Data Card 2
=====
Events (Total) | Rates (Per Second) |
=====
Tx Packets      2959171 | 78799 |
RX Packets      75447396 | 69410 |
-----
Recvd Trigger   0 | 0 |
Collisions     33875 | 846 |
-----
CRC Errors      0 | 0 |
Align Errors    0 | 0 |
-----
Oversize        0 | 0 |
Undersize      49599 | 1213 |
-----
Press ENTER to continue
%
  
```

Remember! Counter Rates are only displayed while transmission is in progress. If you stop the transmission, the rates are always zero.

Troubleshooting Tip: Sometimes you may have a Tcl script that seems to be transmitting and receiving (the TX and RX LEDs are lit), but the received packet counts in your program are always zero. Load LibX and use **set_count** to zero the counters on the receiving card. Run the problem program. Then use **show_count** to display the counters.

If the program has set the packet length to be greater than the maximum length or less than the minimum legal length, the packets will be shown as undersize or oversize, *not* received packets. If the transmit card was set to transmit CRC errors, the packets will be shown under CRC errors, etc.

Procedure Defaults

There are optional defaults on many LibX procedures. The summary table at the end of this chapter shows the syntax of all LibX commands.

The burst command is shown as:

```
burst GROUP <burst size 100>
```

The command name is **burst**. As we have seen, the GROUP argument is the target card entered as \$card1 or \$card2, etc. The third argument in angle brackets, here **burst size**, is an optional argument. If you enter:

```
burst $card1
```

...card 1 will transmit 100 packets. The default value of 100 is used.

If you enter:

```
burst $card1 10000
```

...card1 will transmit 10000 packets. The value 10000 overrides the default value.

If you look at the first line of the **burst** procedure in libx.tcl, it looks like this:

```
proc burst { {group} {burst_size 100} } {
```

The argument **burst_size** and the default value **100** are enclosed in curly braces. Arguments that are not optional, such as **group**, do not have a default value. If you call **burst** without the **group** argument, you will get an error.

Try It! – Procedure Defaults

- Clear the counters on both cards.
- Send a 100-packet burst using the default value.
- Display the counters to verify that 100 packets were sent.
- Send a 5000-packet burst by specifying the burst count.
- Display the counter data to verify that 5000 packets were sent this time.

Example 4 – set_capture / show_capture

Counters give you a rough idea of the packets transmitted and received, but in many cases you will want to capture and view the packet data. This is especially true if you are setting up a program and want to be sure of the packets you think you are sending.

The LibX capture commands will work on all capture-capable cards, including the SX-7210/7410, L3-6710, ML-7710, and Gigabit cards.

set_capture

Unlike counters, which are “running” all the time, capture must be started first. There are varying options for each card capture type. The LibX capture procedures set them all the same:

- Capture all packets
- Capture complete packets
- Capture as much as possible.

Accordingly, the usage is the same for all cards: **set_capture GROUP**.

show_capture

This command stops the currently running capture and displays the specified number of packets, formatted in the traditional 16-bytes-per-line hex dump.

Checking the syntax in the summary table (end of chapter), you will find:

```
show_capture GROUP <NUM> <output>
```

—where NUM is the number of packets to display (default is 5), and output is an optional output file. All LibX **show** commands have the option of redirecting the output to a file rather than to the display.

Writing Output to a File

If you look at the first line of the **show_capture** procedure, you will see:

```
proc show_capture { {group} {CAP_COUNT 5} {output stdout} } {
```

All the **puts** statements in the procedure specify **\$output** as the output target. For example:

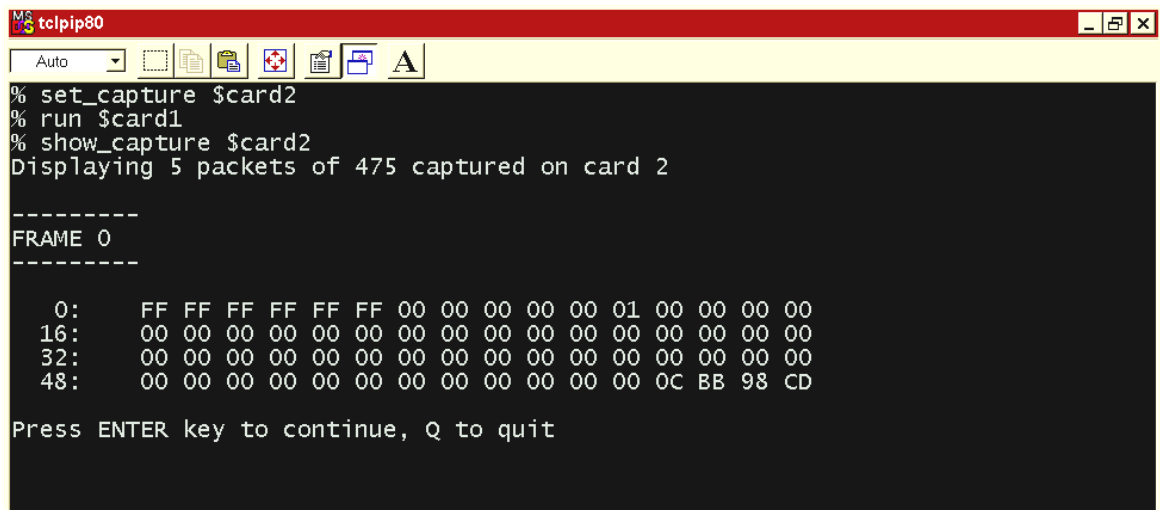
```
puts $output "No packets captured on card [expr $$ + 1]"
```

Since output is defaulted to **stdout** (the display), if you don't specify a value for output, the output goes to the screen (as always). If you do want the output redirected, the calling program opens a file and passes in the handle as the argument to output. Now all **puts** commands will send the output to the file. To be explicit: If you wanted to send the first 100 packets captured to a file named **capture.log**, you would use a sequence like this:

```
set fileID [open capture.log a]
set_capture $card2
run $card1
after 1000
show_capture $card2 100 $fileID
```

Try It! – set_capture / show_capture

- Ensure the system is ready for LibX commands.
- Connect card 1 and card 2 back to back with a crossover cable and ensure the cards are linked.
- Start a capture on card 2 with **set_capture \$card2**.
- Start transmitting from card1.
- Stop and display the capture on card 2 with **show_capture \$card2**.



```
MS tcpip80
Auto
% set_capture $card2
% run $card1
% show_capture $card2
Displaying 5 packets of 475 captured on card 2

-----
FRAME 0
-----

  0:  FF FF FF FF FF FF 00 00 00 00 00 01 00 00 00 00
 16:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 32:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 48:  00 00 00 00 00 00 00 00 00 00 00 00 0C BB 98 CD

Press ENTER key to continue, Q to quit
```

Notice the packet contents are not all zero, as they are on power up. The source and destination MAC addresses were set by the **set_default** procedure. The last four non-zero bytes are the packet CRC.

If you request a larger number to display than are in the capture buffer, the procedure will adjust the number to display to the actual number in the capture buffer.

Capture Problems

It seems obvious that if you want to capture the packets transmitted from card 1, you should start the capture on another card (and not on card 1), but this is a common mistake.

By default, SmartMetrics cards do not capture ARP packets. The L3-6710 card has never done this, because it doesn't have sufficient processing power to capture and respond to ARPs at the same time. The ML-7710 card has enough processing power (since it has two separate processors) and now has the capability in firmware, but the function is not yet part of the Programming Library.

Example 5 – set_mii / show_mii

Two LibX procedures read and write the MII registers on 10/100 cards. These procedures will work on any NetcomSystems 10/100 Ethernet card (including the SX-7405, SX-7205, SX-7410, SX-7210, and ML-7710).

MII Overview

Manipulating the values in the MII registers controls the speed and duplex mode of the FastCard. We will discuss the first six registers:

- **Register 0** Control register (READ/WRITE). Controls the configuration of the local PHY or physical interface.
- **Register 1** Status register (READ ONLY). Shows the status of the local PHY.
- **Register 2 and 3** ID registers (READ ONLY). Shows the make and model of the PHY.
- **Register 4** Advertisement (READ/WRITE). Sets the speed and duplex modes we will accept during the autonegotiation process.
- **Register 5** Link Partner (READ ONLY). Sets the speed and duplex mode the device on the other end of the link is willing to accept.

Essentially, there are two ways to set the speed and duplex mode on a device.

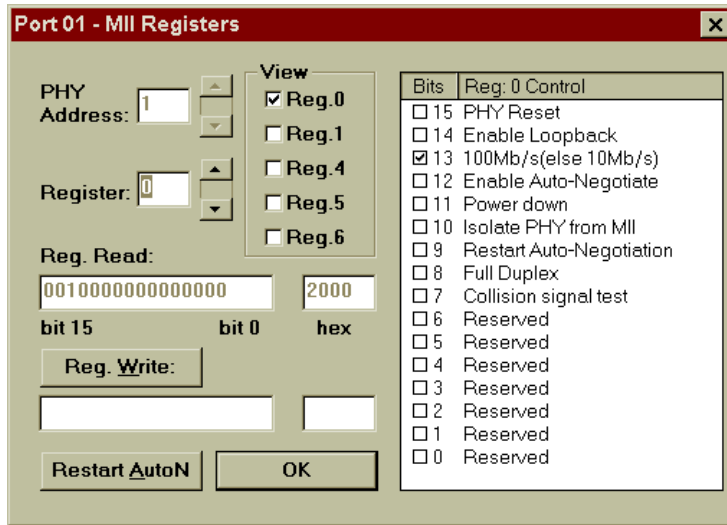
1. Disable autonegotiation by clearing the Enable Autonegotiation bit in the Control Register, then forcing the speed and duplex mode by setting the corresponding bits on the Control Register.
2. Enable autonegotiation by setting the Enable Autonegotiation bit in the Control Register. The link speed and duplex mode will then be negotiated by the two devices on the link. The outcome will be the highest common mode on Register 4 (what we are advertising) and Register 5 (what the other side is advertising).

The Control Register

The Control Register is where we enable and disable autonegotiation, restart autonegotiation, and where, if autonegotiation is disabled, we set the speed and duplex mode for the local device.

The bit position identification for the MII Control Register is shown in the SmartWindow MII window (see below). The most significant bits are as follows:

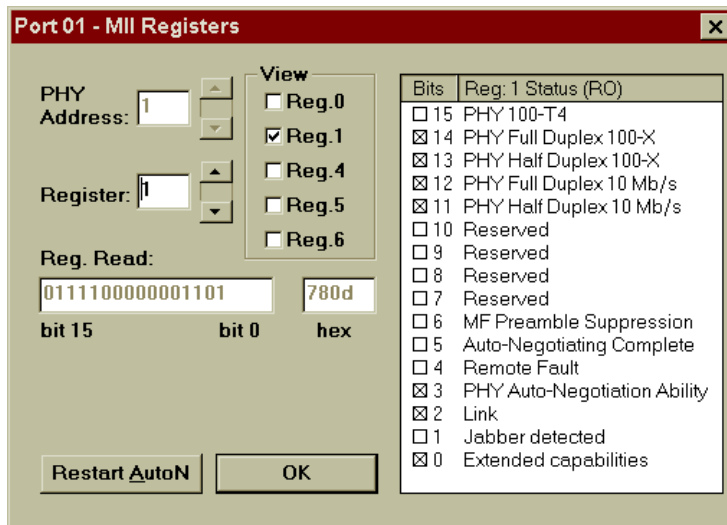
Control Register – Most Significant Bits	
Bit	Description
12	Enables and disables autonegotiation.
9	Restarts autonegotiation.
13	Sets the speed for the local device if bit 12 is cleared (autonegotiation disabled).
8	Sets the duplex mode for the local device if bit 12 is cleared.



The Status Register

The bit position identification for the MII Status Register also is shown in the SmartWindow MII window (see below). The most significant bits are as follows:

Status Register – Most Significant Bits	
Bit	Description
11 – 14	Modes supported by this PHY.
5	Autonegotiation Complete. Indicates autonegotiation cycle is complete.
3	Indicates this PHY is Autonegotiation capable.
2	Indicates the link is up.



Try It! – set_mii / show_mii

- Ensure the system is ready to accept LibX commands (shell started, library loaded, etc).
- Enter **show_mii \$card1**. The current state of the first 6 MII registers is displayed, as shown below:

```

MS-DOS tc!pip80
Auto
% show_mii $card1
*****
Reading MII Registers at Address 1 on card 1
*****
Register 0 Control      ->   : 3100
Register 1 Status      ->   : 7829
Register 2 PHY Identifier ->   : 7810
Register 3 PHY Identifier ->   : 0001
Register 4 Advertisement ->   : 01e1
Register 5 Link Partner ->   : 0081
*****
Press ENTER to continue
_

```

Q. The current value of the Control Register is 0x3100. What does this indicate?

A. The bits for 100Mb (0x2000), Enable Autonegotiation (0x1000), and Full Duplex Mode (0x0100) are set.

Q. Does this setting mean the device is linked at 100Mb Full Duplex Mode?

A. No. The speed and duplex settings in the Control Register have meaning only if Autonegotiation is disabled. When Autonegotiation is enabled in the Control Register, the speed and duplex mode are determined by negotiation between the two devices on the link.

Q. What value would we need to write as the Control Word to force the local device to 10Mb, Half Duplex

A. 0x0000. This would clear the 100Mb and Full Duplex bits and Disable Autonegotiation

Let's try it. First look at the summary table for the syntax of the **set_mii** command. It is shown as:

```
set_mii GROUP <WORD> <REG>
```

We should be pretty familiar with the command GROUP part of a LibX command by now. WORD is the control word we want to write to the register, and REG is the number of the register we want to write to.

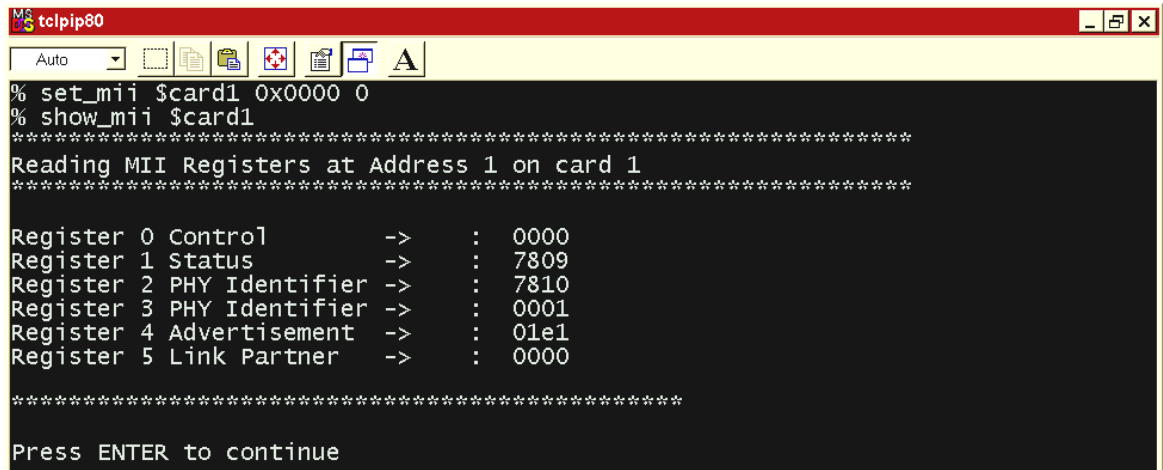
So to send the control word 0x0000 to register 0, we would use **set_mii \$card1 0x0000 0**

- Use the **set_mii** command to force card1 to 10Mb Half Duplex.
- Use **show_mii** to display the register settings.

The result should be as shown below. This will also probably drop the link unless the other side happens to be at 10Mb Half Duplex. If you look at the bit pattern of the status register, you will see the Link bit is no longer set.

- Enable Autonegotiation and force an autonegotiation by writing 0x1200 to the control register (0x1000 = Enable Autonegotiation, 0X0200 = Restart Autonegotiation).

The cards should relink.



```
MS tclpip80
Auto
% set_mii $card1 0x0000 0
% show_mii $card1
*****
Reading MII Registers at Address 1 on card 1
*****
Register 0 Control      ->  : 0000
Register 1 Status      ->  : 7809
Register 2 PHY Identifier ->  : 7810
Register 3 PHY Identifier ->  : 0001
Register 4 Advertisement ->  : 01e1
Register 5 Link Partner ->  : 0000
*****
Press ENTER to continue
```

MII Write Problems

Note that the control word is formatted with a leading 0x. This indicates the value is in hex. If you do not use the leading 0x, the value you enter will be interpreted as decimal, probably *not* what you intended. When you read it back with **show_mii**, it will be displayed in hex, which might leave you wondering “What happened?” Now you know.

Not all fields on the READ/WRITE registers on all transceivers are writeable. If you send a 7 you may read back a 5 (assuming the 2 bit is not writeable).

Example 6 – set_I3 / show_I3

The `set_I3 / show_I3` commands set, configure, and display the Layer 3 settings for a target SmartMetrics card.

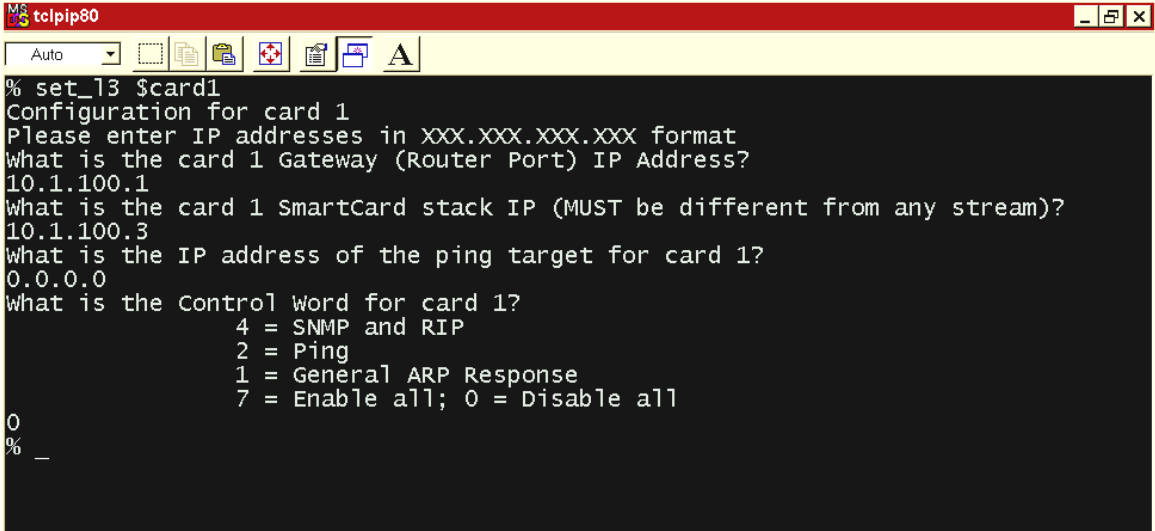
These settings set the MAC address and IP address of the card stack, specify the default gateway, and configure, enable, or disable background packet generation (PING, RIP, and ICMP).

set_I3

The `set_I3` LibX function will prompt for the most significant Layer 3 settings. You enter the parameters when prompted and the card configuration is set.

Try It! – set_I3

- Ensure the system is set to receive LibX commands.
- Run the `set_I3` procedure for card1 by entering `set_I3 $card1`.
- Enter configuration values when prompted, as shown below.



```
MS tcPIP80
Auto
% set_I3 $card1
Configuration for card 1
Please enter IP addresses in XXX.XXX.XXX.XXX format
What is the card 1 Gateway (Router Port) IP Address?
10.1.100.1
What is the card 1 SmartCard stack IP (MUST be different from any stream)?
10.1.100.3
What is the IP address of the ping target for card 1?
0.0.0.0
What is the Control Word for card 1?
    4 = SNMP and RIP
    2 = Ping
    1 = General ARP Response
    7 = Enable all; 0 = Disable all
0
% _
```

In the example above, we set the following values:

- Default gateway address (the IP address of the router port you are connected to).
- Card stack IP address (normally in the same subnet as the router port).
- IP address of the PING target (if PINGs were enabled this would be the address we would be PING-ing for).
- Control word.

Bits 2 and 4 enable and disable background packet generation. Bit 1 will cause the card to replay to ALL ARPs on the network and should never be set on a live network.

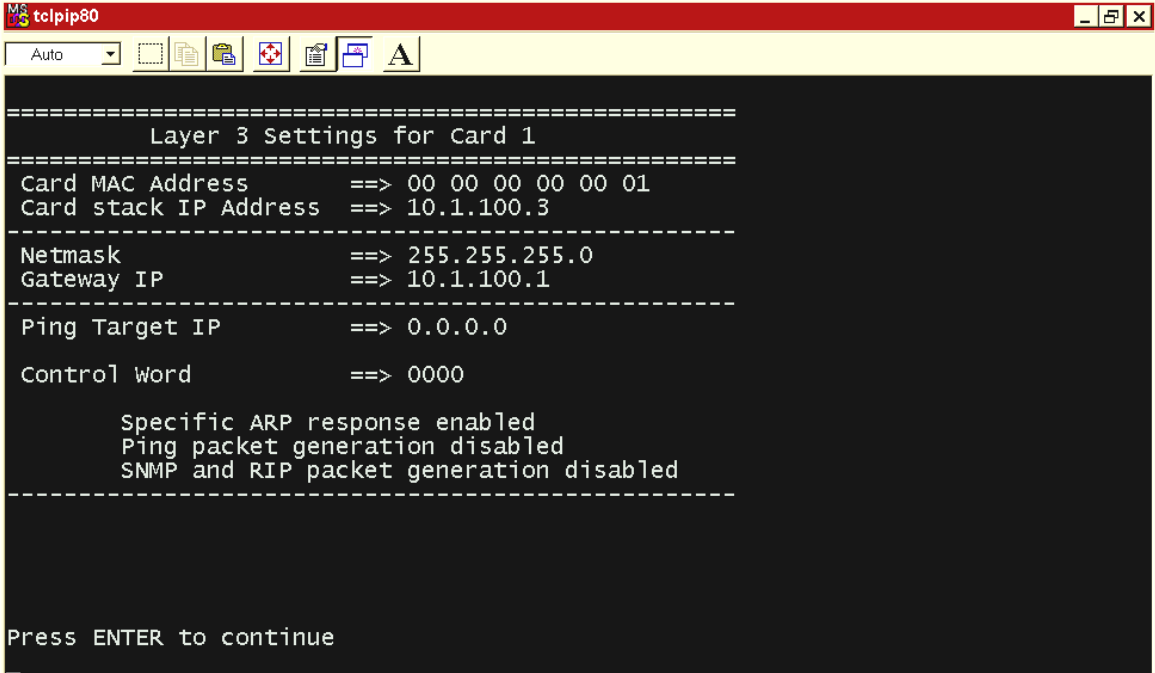
show_l3

The companion command is **show_l3**. This command will display the current setting of the most significant L3 settings on the target card.

Try It! – show_l3

- To display the Layer 3 settings for card 1, enter `show_l3 $card1`.

The Layer 3 settings are displayed with formatting as shown below. Note that the user was not prompted by **set_l3** for some fields displayed here, such as Netmask and Card MAC. If necessary, the program can be modified by the end user to prompt for fields specific to his/her own requirements.



```
MS tcpip80
Auto
=====
Layer 3 Settings for Card 1
=====
Card MAC Address      ==> 00 00 00 00 00 01
Card stack IP Address ==> 10.1.100.3
-----
Netmask               ==> 255.255.255.0
Gateway IP            ==> 10.1.100.1
-----
Ping Target IP        ==> 0.0.0.0
Control Word          ==> 0000

    Specific ARP response enabled
    Ping packet generation disabled
    SNMP and RIP packet generation disabled
-----

Press ENTER to continue
```

Example 7 – set_streamxx / show_stream

Streams are the core of the SmartMetrics cards. They allow you to set up over 1000 different VTEs or streams per card, with control over the contents of every protocol header field. The trick is to set up all those streams without making a mistake, such as having the same IP address for more than one MAC address, duplicating IP addresses, or setting the wrong streams to the wrong gateway.

set_streamip

The LibX **set_stream** commands automate the stream creation process and will usually allow you to set up streams without duplications or mismatches.

There are three stream-creation commands: **set_streamip**, **set_streamudp**, and **set_streamipx**, used to create groups of IP, UDP, or IPX packets, respectively.

If we look at summary table for the syntax of the **set_streamip** command, we see:

```
set_streamip GRP1 GRP2 <NUM 10> <sourceIP> <destIP>
```

The arguments are as follows:

Argument	Description
GRP1	Transmitting card or group of cards
GRP2	Receiving card or group of cards.
NUM	Number of streams to create.
sourceIP	Source IP address of the first stream.
destIP	Destination IP address of the first stream.

Let's look at the default case first. If we enter **set_streamip \$card1 \$card2**, we will set up ten streams on card1. The stream creation commands require two GROUP or card arguments, because we use the card numbers to generate unique stream numbers.

The default pattern for IP addresses is 10.X.1.10. When the streams are created, the X is replaced with the card number. So in our example, if we are transmitting from card 1 to card 2, the source IP would be 10.1.1.10 and the destination IP would be 10.2.1.10.

The card number also replaces the third byte of the MAC address. So the source MAC address for the first stream created on card 2 would be 00 00 00 02 00 01.

Each successive stream created increments the LSB of both MAC address and the least significant octet of both IP addresses.

The MAC addresses and IP addresses for the ten streams created (card1 to card2) would be as shown in the table below.

Stream #	Source IP	Source MAC	Destination IP	Destination MAC
1	10.1.1.10	00 00 00 01 00 01	10.2.1.10	00 00 00 02 00 01
2	10.1.1.11	00 00 00 01 00 02	10.2.1.11	00 00 00 02 00 02
3	10.1.1.12	00 00 00 01 00 03	10.2.1.12	00 00 00 02 00 03
4	10.1.1.13	00 00 00 01 00 04	10.2.1.13	00 00 00 02 00 04
5	10.1.1.14	00 00 00 01 00 05	10.2.1.14	00 00 00 02 00 05
6	10.1.1.15	00 00 00 01 00 06	10.2.1.15	00 00 00 02 00 06
7	10.1.1.16	00 00 00 01 00 07	10.2.1.16	00 00 00 02 00 07
8	10.1.1.17	00 00 00 01 00 08	10.2.1.17	00 00 00 02 00 08
9	10.1.1.18	00 00 00 01 00 09	10.2.1.18	00 00 00 02 00 09
10	10.1.1.19	00 00 00 01 00 10	10.2.1.19	00 00 00 02 00 10

To create a second set of matching streams on card 2, you would enter **set_streamip \$card2 \$card1**

Try It! – set_streamip

- Ensure the system is ready for LibX commands.
- Create the default 10 streams on card 1 with **set_streamip \$card1 \$card2**.
- Create a matching set of streams on card 2 with **set_streamip \$card2 \$card1**.

show_stream

Now that you have created the streams, how can you check them? You could start a capture on one card and transmit from the other and view the captured packets, but decoding raw hex dumps can be tedious. LibX has a command, **show_stream**, that will display the protocol data on the target card, formatted to make it easier to read.

The syntax (from the summary table) is:

```
show_stream GROUP <NUM 5> <output>
```

So by default, it will display the protocol data for the first 5 streams. Like all **show** commands, the output can be redirected to a file (see page 95).

Try It! – show_stream

- After you have created the streams on card 1 and 2 as outlined on the previous page, display the protocol data by entering **show_stream \$card1**.

Notice the MAC and IP addresses substitute the card numbers at the places explained previously. Notice also the frame length and ARP status for this stream is displayed below the IP addresses.

- Display the protocol data for card 2 using the **show_stream** command. Note again how the default-addressing scheme replaces the indicated values with the card number.


```
MS tclpip80
Auto
Card 1 - Stream 1 of 10 - STATUS => Active
-----
Layer 2 Data:
  Destination MAC ==> 00 00 00 02 00 01
  Source MAC      ==> 00 00 00 01 00 01
-----
Layer 3 Data:
  Stream type is IP
  Destination IP  ==> 10.2.1.10
  Source IP       ==> 10.1.1.10

  Frame length is 100 bytes
  ARP requests have not been sent
-----
                    Raw Protocol Data
-----
00:  00 00 00 02 00 01 00 00 00 01 00 01 00 0A 00 00
10:  0A 02 01 0A 0A 01 01 0A FF FF FF 00 0A 01 01 01
20:  04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Press ENTER to continue
```

Adding Additional Streams and Removing Streams

To add additional streams, simply run the desired stream creation command again. The `set_stream` commands will append new streams to those already on the card.

If you want to remove all the streams on the card you can:

- Use a `set_stream` command passing zero as the number of streams—for example, `set_streamip $card1 $card2 0`.
- Use the LibX `set_default` procedure. The default for SmartMetrics cards is zero streams.
- Use the `set_I2` LibX command (covered in the next section).
- Use the library `HTResetPort` command (a reset erases all streams).

Non-default IP Addresses

If you do not want the default IP addressing scheme, you can pass in a new IP address. The syntax of the `set_stream` command is:

```
set_streamip GRP1 GRP2 <NUM 10> <sourceIP> <destIP>
```

To set up a group of 10 streams starting with 192.168.X.10 (where X will be replaced with the card number) transmitting to a group of streams starting with 13.X.27.20, you would enter `set_streamip $card1 $card2 10 192.168.X.10 13.X.27.20`.

Note that if you want to override a default value, you have to set all optional values to the left of that value. In this case for example, we have to explicitly pass in the NUM value of 10, even though that is the default value, if we want to change the IP addresses.

Many to One and One to Many

You can take advantage of LibX's ability to accept groups as arguments to specify test setups of one to many, many to one, or many to many.

As covered previously, it is possible to set a group of cards using the form:

```
set RxGroup [list $card2 $card3 $card4]
```

This creates a virtual group of three cards that can be passed into any LibX command. For example, after creating the RxGroup we could start capture on all three by entering **set_capture \$RxGroup**.

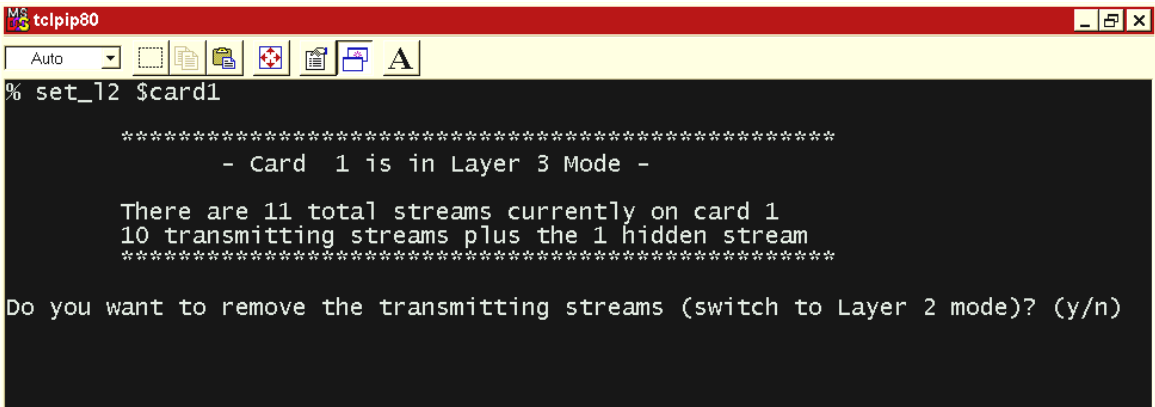
We can use the same capability to do many-to-one or one-to-many tests.

For example, **set_streamudp \$card1 \$RxGroup** will set up 30 streams on card 1, 10 addressed for card 2, 10 addressed for card 3, and 10 addressed for card 4.

Example 8 – Other Stream Commands

There are two additional stream commands to know:

- **set_l2** Displays the number of streams on the card and gives the user the opportunity to erase the streams and switch to layer 2 (Traditional) mode.
- **set_arp** Sends ARP requests from all active streams on the card.



```
MS tcpip80
Auto
% set_l2 $card1
*****
- Card 1 is in Layer 3 Mode -
*****
There are 11 total streams currently on card 1
10 transmitting streams plus the 1 hidden stream
*****
Do you want to remove the transmitting streams (switch to Layer 2 mode)? (y/n)
```

set_l2

Erasing all streams puts the card in Layer 2 (Traditional or L2) mode. The syntax is **set_l2 GROUP**.

set_arp

Sends ARP requests from all active streams. Remember that the **show_stream** command also displays ARP status. You can use that to check that the ARP cycles are complete. The syntax is **set_arp GROUP**

Summary of LibX Procedures

Procedure	Arguments	Description
run	GROUP	Starts card transmitting (continuous mode).
stop	GROUP	Stops transmitting.
burst	GROUP <burst size 100>	Sends a burst of packets (restores to continuous mode on exit).
set_link	–	Checks for library file installation and current link. If not linked, prompts for COM port or Ethernet address.
check_link	–	Checks for link. Calls <code>set_link</code> if not linked.
set_count	GROUP	Resets counters with HTCclearPort.
show_count	GROUP <output>	Displays standard counter data. Output option allows file handle to be used as output target (write to a file rather than <code>stdout</code>).
set_capture	GROUP	Resets counters to capture all. Works on Fast Ethernet and L3 cards.
show_capture	GROUP <NUM> <output>	Displays number of packets set by NUM. Default is 5 packets. Works with both L3 and Fast Ethernet cards. Output option same as for <code>show_count</code> .
set_mii	GROUP <WORD> <REG>	Sets register REG with the value of WORD. Defaults are 0 and 0x0000.
show_mii	GROUP <output>	Shows current contents of first 5 MII registers. Output option same as for <code>show_count</code> .
check_cardlink	CARD	Checks the card's current link state. Returns 1 if the link is up, -1 if the link is down.
set_streamtcp	GRP1 GRP2 <NUM 10> <sourceIP> <destIP>	Sets up NUM TCP streams on the card(s) in the GRP1 list (default 10). It uses the source and destination CardList Elements to generate the streams where the IP addresses are 10.x.1.10 (X = number of card) and MAC addresses of 00 00 00 XX 00 01 (XX = card number).
set_default	GROUP	Sets target card to reasonable default values. Works on Fast Ethernet, L3, Gigabit, and ATM cards. Defaults are generally those of SmartWindow.
set_streamip	GRP1 GRP2 <NUM 10>	Similar to <code>set_streamtcp</code> above.
set_streamudp	GRP1 GRP2 <NUM 10>	Similar to <code>set_streamtcp</code> above.
set_streamipx	GRP1 GRP2 <NUM 10>	Similar to above but uses a similar scheme to generate Host and Network numbers.
show_stream	GROUP <NUM 5> <output>	Shows streams on the target card. Breaks out fundamental information such as protocol, source and destination IP addresses, source and destination MAC addresses, and length. Shows raw data dump under protocol breakout. Output option same as <code>show_count</code> .
set_l3	GROUP	Interactive utility for configuring L3 parameters. Prompts user for gateway, SmartCard, and PING destination IP addresses and Control word.
show_l3	GROUP <output>	Shows L3 configuration. Output options same as for <code>show_count</code> .
set_l2	GROUP	Displays the number of transmitting streams on target card and allows the user to remove them, setting the card in L2 mode.

Chapter 12:

Function and Structure Reference

This chapter describes the SmartLib Original Functions in alphabetical order.

NOTE Certain functions require a long time to execute. This is particularly true of the VFD and Capture-related functions when they are passing large quantities of data.

Function Prefixes: ET, HT, HG, NS

Function prefixes identify the related system hardware, as follows:

- Functions prefixed with **ET** apply to SmartBits chassis (originally, to the ET-1000).
- Functions prefixed with **HT** apply to a single port. These functions require a Hub/Slot/Port triple in the parameter list.
- Functions prefixed with **HG** operate on a group of ports, as defined in a string passed to the `HGSetGroup(PortIdGroup)` command. A port group can be maintained and modified by using the following commands:
 - ❖ `int HGAddtoGroup(iHub,iSlot,iPort)`
 - ❖ `int HGRemoveFromGroup()`
 - ❖ `int HGRemovePortIdFromGroup()`
 - ❖ `int HGIsPortInGroup()`
 - ❖ `int HGIsHubSlotPortInGroup()`
 - ❖ `int HGGetGroupCount()`
- Functions prefixed with **NS** can communicate with a SmartBits controller or perform a general action for SmartCards or modules, such as create a frame template.

See the detailed command descriptions that follow.

Data Structures

When an Original Function uses a data structure, the structure is described here just following the function. If a data structure is used by different functions, it is simply repeated for each function and so may appear more than once in this guide.

NOTE For complete information on the structures used by the `SetStructure` and the `GetStructure` commands, refer to the *Message Functions* manual.

Usage

Some data structures require additional memory allocation. In most cases, you define the structure at the beginning of your function. For example:

```
int SetETCollision(void)
{
    CollisionStructure Collide;    //Collision structure
    Collide.Offset = 0x20;
    Collide.Duration = 0x36;
    Collide.Count = 14486;
    Collide.Mode = CORP_A;
    ETCollision(&Collide); //Set it so
}
```

Some library functions automatically put information into the structures you declare. In these cases, declare the functions and then call the appropriate library routine. For example:

```
int GetETCollision(void)
{
    CollisionStructure Collide; //defines a structure
    ETGetCollision(&Collide);  //which the library fills
    printf("Collision Offset is:  %d\n", Collide.Offset);
    printf("Collision Duration is: %d\n", Collide.Duration);
}
```

Some library functions require you to put information into the declared data structures before calling them. If this is not done, the library might produce unpredictable results. For example:

```
int BadSetETCollision(void)
{
    CollisionStructure Collide; //defines a structure, but
    //contents unspecified
    ETCollision(&Collide);    //call with unintended
    //results
}
```

ETEnableBackgroundProcessing

Description	Allows enhanced responsiveness of foreground applications.
Syntax	int ETEnableBackgroundProcessing(int bFlag)
Parameters	<i>bFlag</i> int 0 to disallow, 1 to allow.
Return Value	The return value is the previous state of BackgroundProcessing.
Comments	<p>Use this function with extreme care. All commands to the Programming library are executed completely then returned. ETEnableBackgroundProcessing allows for the same process or other processes to proceed while a Programming library function is being executed. A guard flag is enabled around reentrancy in the library, but you could end up in “deadly-embrace” situations. If this function is enabled, while a command in the Programming Library is executing, you are performing operations on the stack. So, do not use WM_TIMER messages, or button press messages to call Programming Library functions if this function is enabled. The code executed when background processing is enabled is below. Note the PeekMessage loop does not process WM_USER+n messages.</p> <pre> if (bAllowIdleProcessing) { bldling = TRUE; while(PeekMessage(&Msg,NULL,WM_NULL,WM_USER-1,PM_REMOVE)) { TranslateMessage(&Msg); DispatchMessage(&Msg); } } bldling = FALSE; </pre>

ETGetBaud

Description	Obtain the current baud rate settings for the communications port.
Syntax	long ETGetBaud(void)
Parameters	None
Return Value	The return value indicates the baud rate as a long value.
Comments	None

ETGetController

Description	Returns the current type of SMB controller.												
Syntax	int ETGetController(void)												
Parameters	None												
Return Value	<table> <tr><td>1</td><td>CONTROLLER_ET1000</td></tr> <tr><td>2</td><td>CONTROLLER_SMB1000</td></tr> <tr><td>3</td><td>CONTROLLER_SMB2000</td></tr> <tr><td>4</td><td>CONTROLLER_SMB6000</td></tr> <tr><td>5</td><td>CONTROLLER_SMB200</td></tr> <tr><td>6</td><td>CONTROLLER_SMB600</td></tr> </table>	1	CONTROLLER_ET1000	2	CONTROLLER_SMB1000	3	CONTROLLER_SMB2000	4	CONTROLLER_SMB6000	5	CONTROLLER_SMB200	6	CONTROLLER_SMB600
1	CONTROLLER_ET1000												
2	CONTROLLER_SMB1000												
3	CONTROLLER_SMB2000												
4	CONTROLLER_SMB6000												
5	CONTROLLER_SMB200												
6	CONTROLLER_SMB600												
Comments													

ETGetFirmwareVersion

Description	Retrieves the current SmartBits firmware version of the attached SmartBits. It is expressed as an eight-character array with a terminating NULL character, which is left in buffer. Buffer must have enough room for at least 9 characters.
Syntax	int ETGetFirmwareVersion(char* Buffer)
Parameters	<i>Buffer</i> char* Points to a memory location where the version information is to be placed. NOTE: Buffer must be at least 9 characters long.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if there was a failure. <i>See Appendix B for error codes.</i>
Comments	The version is returned as a character string, not an integer.

ETGetHardwareVersion

Description	Retrieves the current hardware version of the attached SmartBits. It is expressed as an eight-character array with a terminating NULL character, which is left in Buffer. Buffer must have enough room for at least 9 characters.
Syntax	int ETGetHardwareVersion(char* Buffer)
Parameters	<i>Buffer</i> char* Points to a memory location where the version information is to be placed. NOTE: Buffer must be at least 9 characters wide.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if there was a failure. <i>See Appendix B for error codes.</i>
Comments	The version is returned as a character string, not an integer.

ETGetLibVersion

Description	Used to retrieve the version information for the Programming Library currently in use by the program making the call. The first string is a text description of the library. The second string is the version number in ASCII.
Syntax	int ETGetLibVersion(char* pszDescription, char* pszVersion)
Parameters	<i>pszDescription</i> char* Points to a memory location where the library description is to be placed. NOTE: Buffer must be at least 50 characters wide. <i>pszVersion</i> char* Points to a memory location where the version information is to be placed. NOTE: Buffer must be at least 20 characters wide.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if there was a failure. <i>See Appendix B for error codes.</i>
Comments	The version is returned as a character string, not an integer.

ETGetLinkFromIndex

Description	Returns the SmartBits ComPort.
Syntax	int ETGetLinkFromIndex(int iLink)
Parameters	<i>iLink</i> int Specifies which SmartBits connection. A value of 1 means the first SmartBits connection to the Programming Library.
Return Value	This function returns the SmartBits ComPort which is associated with the specified ETLink attempt. The return value is < 0 if there was a failure. <i>See Appendix B for error codes.</i>
Comments	See ETSetCurrentLink.

ETGetLinkStatus

Description	Indicates the current status of the link between the PC and the attached SmartBits.
Syntax	int ETGetLinkStatus(void)
Parameters	None
Return Value	Returns the identity of the COM port if the link is established. Returns a failure code if the function failed. <i>See Appendix B for error codes.</i>
Comments	Use this function to determine whether or not there is a communication link established with an attached SmartBits. If the link has already been established and then is abruptly broken (due to a physical break in the connecting device or cable), this function will return a 0.

ETGetSerialNumber

Description	Retrieves the current serial number of the attached SmartBits. It is expressed as an eight-character array with a terminating NULL character, which is left in Buffer. Buffer must have enough room for at least 9 characters.
Syntax	int ETGetSerialNumber(char* Buffer)
Parameters	<i>Buffer</i> char* Points to a memory location where the serial number is to be placed. NOTE: Buffer must be at least 9 characters wide.
Return Value	The return value is >= 0 if the function executed successfully. The return value is < 0 if there was a failure. <i>See Appendix B for error codes.</i>
Comments	The serial number is returned as a character string, not an integer.

ETGetTotalLinks

Description	Returns total SmartBits connections.
Syntax	int ETGetTotalLinks(void)
Parameters	None
Return Value	This function returns the total SmartBits systems connected to Programming Library. A value of 2 means there are two SmartBits connected.
Comments	See ETSetCurrentLink.

ETIsBackgroundProcessing

Description	Determines if the Programming Library is currently executing a function.
Syntax	int ETIsBackgroundProcessing(void)
Parameters	None
Return Value	The return value is >0 if true, 0 if false. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This returns the state of the guard flag used to control reentrancy in the Programming Library.

ETLink

Description	Forges a communication link between the PC and the attached SmartBits.								
Syntax	int ETLink(int ComPort)								
Parameters	<p><i>ComPort</i> int Determines the COM port to be used to run the remote link to the attached SmartBits:</p> <table style="margin-left: 40px;"> <tr> <td>ETCOM1</td> <td>Serial COM port 1</td> </tr> <tr> <td>ETCOM2</td> <td>Serial COM port 2</td> </tr> <tr> <td>ETCOM3</td> <td>Serial COM port 3</td> </tr> <tr> <td>ETCOM4</td> <td>Serial COM port 4</td> </tr> </table> <p>Any <i>ComPort</i> values outside this range are discarded and will have no effect on the link status.</p>	ETCOM1	Serial COM port 1	ETCOM2	Serial COM port 2	ETCOM3	Serial COM port 3	ETCOM4	Serial COM port 4
ETCOM1	Serial COM port 1								
ETCOM2	Serial COM port 2								
ETCOM3	Serial COM port 3								
ETCOM4	Serial COM port 4								
Return Value	The return value is less than or equal to 0 if the function failed to establish a link with the attached SmartBits.								
Comments	This function must execute successfully before any communication between the host PC and the remote SmartBits can take place. While executing this function, the PC will search for the baud rate at which the attached SmartBits responds. It may take a while (up to 30 seconds) for this function to execute, as it must seek out and search several baud rates before deciding whether or not the attached SmartBits is responding correctly.								

ETMake2DArray

Description	Creates virtual two-dimensional arrays with the Tcl programming language.
Syntax	int ETMake2DArray (char* pszArrayName, int iSizeFirstDim, int iSizeSecondDim)
Parameters	<p><i>pszArrayName</i> char* A pointer to the name of the virtual array created with Tcl. Use <i>pszArrayName</i> for any functions that require 2D arrays.</p> <p><i>iSizeFirstDim</i> int Specifies the number of elements in the first dimension of the array.</p> <p><i>iSizeSecondDim</i> int Specifies the number of elements in the second dimension of the array.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if this function failed to execute. <i>See Appendix B for error codes.</i>
Comments	<p>This is a Tcl work-around, only found in ET1000.TCL.</p> <p>This Tcl utility function can be used, for example, with HTCARDModels where the first array is MAX_HUBS and the second array is MAX_SLOTS.</p> <p>For more information, see <i>Tcl_tips.txt</i> in your SmartLib installation.</p>

ETMake3DArray

Description	Creates virtual three-dimensional arrays with the Tcl programming language.
Syntax	int ETMake3DArray (char* pszArrayName, int iSizeFirstDim, int iSizeSecondDim, int iSizeThirdDim)
Parameters	<p><i>pszArrayName</i> char* A pointer to the name of the virtual array created with Tcl. Use pszArrayName for any functions that require 3D arrays.</p> <p><i>iSizeFirstDim</i> int Specifies the number of elements in the first dimension of the array.</p> <p><i>iSizeSecondDim</i> int Specifies the number of elements in the second dimension of the array.</p> <p><i>iSizeThirdDim</i> int Specifies the number of elements in the third dimension of the array.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if this function failed to execute. <i>See Appendix B for error codes.</i>
Comments	<p>This is a Tcl work-around, only found in ET1000.TCL.</p> <p>This Tcl utility function can be used, for example, with HTFrame where the first array is iHub, the second is iSlot, and the third is iPort.</p> <p>For more information, see Tcl_tips.txt in your SmartLib installation.</p>

ETSetBaud

Description	Adjusts the baud rate of the SmartBits's serial link.										
Syntax	int ETSetBaud(int Baud)										
Parameters	<p><i>Baud</i> int Determines the Baud rate at which the attached SmartBits operates:</p> <table style="margin-left: 40px;"> <tr><td>ETBAUD2400</td><td>2400 Baud</td></tr> <tr><td>ETBAUD4800</td><td>4800 Baud</td></tr> <tr><td>ETBAUD9600</td><td>9600 Baud</td></tr> <tr><td>ETBAUD19200</td><td>19.2 kBaud</td></tr> <tr><td>ETBAUD38400</td><td>38.4 kBaud</td></tr> </table> <p style="text-align: right;">All other values are invalid and will have no effect on the attached SmartBits.</p>	ETBAUD2400	2400 Baud	ETBAUD4800	4800 Baud	ETBAUD9600	9600 Baud	ETBAUD19200	19.2 kBaud	ETBAUD38400	38.4 kBaud
ETBAUD2400	2400 Baud										
ETBAUD4800	4800 Baud										
ETBAUD9600	9600 Baud										
ETBAUD19200	19.2 kBaud										
ETBAUD38400	38.4 kBaud										
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>										
Comments	<p>Once the Baud rate of the attached SmartBits has been changed, it will no longer be able to communicate with the PC. After executing this function, you should break and re-establish the link using the ETUnLink and ETLink functions. (The ETLink function automatically finds the Baud rate at which the attached SmartBits is currently operating.) ADVICE: If problems occur while trying to link at a different baud rate, place the SmartBits in the local mode by pressing its RESET switch. Then activate mode A4 and SET the baud rate as appropriate.</p>										

ETSetCurrentLink

Description	Specifies which SmartLib Link (SMB to PC) is the current Link. If you have multiple Links, use this command before sending ET controller-specific commands such as ETGetHardwareVersion. You need not use this command before sending SmartCard-specific commands.										
Syntax	int ETSetCurrentLink(int ComPort)										
Parameters	<table border="0"> <tr> <td><i>ComPort</i></td> <td>int Specified the attached SmartBits with <i>ComPort</i> to be used in SmartLib for related ET commands:</td> </tr> <tr> <td>ETCOM1</td> <td>Serial COM port 1</td> </tr> <tr> <td>ETCOM2</td> <td>Serial COM port 2</td> </tr> <tr> <td>ETCOM3</td> <td>Serial COM port 3</td> </tr> <tr> <td>ETCOM4</td> <td>Serial COM port 4</td> </tr> </table> <p>Any <i>ComPort</i> values outside this range are discarded and will have no effect on the link status.</p>	<i>ComPort</i>	int Specified the attached SmartBits with <i>ComPort</i> to be used in SmartLib for related ET commands:	ETCOM1	Serial COM port 1	ETCOM2	Serial COM port 2	ETCOM3	Serial COM port 3	ETCOM4	Serial COM port 4
<i>ComPort</i>	int Specified the attached SmartBits with <i>ComPort</i> to be used in SmartLib for related ET commands:										
ETCOM1	Serial COM port 1										
ETCOM2	Serial COM port 2										
ETCOM3	Serial COM port 3										
ETCOM4	Serial COM port 4										
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>										
Comments	Instead of changing ET related commands, to include another parameter to specify which SmartBits system in the Programming Library functions in order to support multiple SmartBits connections, use ETSetCurrentLink to specify "Current" SmartBits for the related ET commands.										

ETSetCurrentSockLink

Description	Specify which SmartLib link (SMB to PC) is the current link. If you have multiple Links, use this command before sending ET controller-specific commands such as ETGetHardwareVersion. You need not use this command before sending SmartCard-specific commands.
Syntax	int ETSetCurrentSockLink(char* IPAddr)
Parameters	<i>IPAddr</i> char* Specifies the IP address of the SMB controller you want to send a command to.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	

ETSetGPSDelay

Description	Determines the actual start time communicated to a remote hub by HGRun and by HGStart, HGStop, and HGStep when GPS is available. Calculations are based on the estimated time to send a message to the remote hub. The default delay used by HGRun, and HGStart, HGStop, and HGStep for GPS synchronized starts is 20 seconds plus an additional 10 seconds for each hub. Use this function to change the default start time if: * There is not enough time for the remote host to receive the message. This can cause the local hubs to start before the remote hubs receive the command. * The default delay is unnecessarily long.
Syntax	int ETSetGPSDelay(ulong ulSeconds)
Parameters	<i>ulSeconds</i> ulong Determines the delay added to the current time so that local and remote hubs can start synchronously.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command is only used by HGRun, HGStart, HGStop, and HGStep when GPS is available.

ETSetTimeout

Description	Determines how long SmartLib will wait for a response from the SMB controller before timing out. The default timeout value is 5 seconds.
Syntax	int ETSetTimeout(unsigned TimeOutValue)
Parameters	<i>TimeOutValue</i> unsigned int Determines the time-out value, in milliseconds. <i>Range: 1 to 2,147,483,647 milliseconds (0x7FFFFFFF).</i>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Passing a value of 0 will set the timeout to approximately 24 days, effectively disabling timeout for most purposes.

ETSocketLink

Description	Used to connect to a SmartBits system over an IP socket connection. <i>Note: Use a serial port connection first to configure the SmartBits chassis IP address.</i>
Syntax	int ETSocketLink (char IPAddr, int iTCPPort)
Parameters	<i>szIPAddr</i> char Specifies the IP address of the SmartBits chassis to which a connection attempt should be made. <i>iTCPPort</i> int TCP port number. The default value is 16385 .
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code (less than zero) is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Use the serial port interface to set or change the IP address of the SmartBits chassis. To connect the PC to the chassis, use a terminal emulation program such as HyperTerminal. Once connected to SmartBits, enter the command <code>ipaddr</code> to view the current IP address. Enter the command <code>ipaddr<new_address></code> to set a new IP address. For example, enter a command like the following: <code>ipaddr 129.186.145.5</code>

ETUnLink

Description	Causes the communication link between the PC and the attached SmartBits to be broken. The allocated COM port will be freed up for other applications to use.
Syntax	int ETUnLink(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	It is highly recommended that this function be performed as part of shutting down the SmartBits application. This guarantees that DOS will recognize the allocated COM port as having been freed from any application and is thus available. Also, the execution of this function automatically puts the attached SmartBits in the manual mode.

HGAddtoGroup

Description	Along with HGSetGroup, this command can be used to add individual hub/slot/port cards to a group.
Syntax	int HGAddtoGroup (int iHub, int iSlot, int iPort)
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. <i>See Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>Currently, this command should be used in Hub/Slot/Port ascending order. Example:</p> <pre>HGSetGroup (NULL) ; HGAddtoGroup (0, 0, 0) ; HGAddtoGroup (0, 1, 0) ;</pre> <p>This will add the first two cards in the first hub to a group.</p>

HGAlign

Description	Create alignment errors on the previously selected group. This function is valid for SmartCards only.
Syntax	int HGAlign(int iBits)
Parameters	<p><i>iBits</i> int Sets the number of extra alignment bits to transmit. <i>Range:</i> 0 to 7.</p> <p>Setting this value to 0 disables generation of packets with alignment bit errors.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGBurstCount

Description	Sets the number of packets transmitted in a single burst from all ports associated with the PortIdGroup defined by the HGSetGroup(PortIdGroup) command.
Syntax	int HGBurstCount(long lVal)
Parameters	<i>lVal</i> long Specifies the burst count. <i>Range: 1 to 16,777,215.</i>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This instruction does not cause a burst of packets to be sent. Use HGTransmitMode , or HTTransmitMode to select a burst mode, and then use HGRun , HGStart , HGStep , or HTRun to actually start the transmission of the burst.

HGBurstGap

Description	Sets up the time gap between bursts of packets from all ports associated with the PortIdGroup defined by the HGSetGroup(PortIdGroup) command.
Syntax	int HGBurstGap(long lVal)
Parameters	<i>lVal</i> long Specifies the inter-burst gap in tenths of a microsecond. <i>Range: 1 to 16 million.</i>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This instruction is only applied if HGTransmitMode , or HTTransmitMode has selected one of the MULTI_BURST_MODE, or CONTINUOUS_BURST mode selections. Use HGRun , HGStart , and HTRun to actually start the transmission of the bursts.

HGBurstGapAndScale

Description	Sets up the time gap between bursts of packets, at the given scale from all ports associated with the PortIdGroup defined by the HGSetGroup(PortIdGroup) command.
Syntax	int HGBurstGapAndScale(long lVal, int iScale)
Parameters	<i>lVal</i> long Specifies the inter-burst gap value. Legal values range anywhere from the lowest gap possible on the group being addressed up to a maximum of 1.6 sec. <i>iScale</i> int Specifies the scale of the gap value according to following: NANO_SCALE = nanoseconds scale MICRO_SCALE = microseconds scale MILLI_SCALE = milliseconds scale Possible values range from 1 to 1,600,000,000 nanoseconds for NANO_SCALE; from 1 to 1,600,000 microseconds for MICRO_SCALE; and from 1 to 1600 milliseconds for MILLI_SCALE. Values outside this range return an error code.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This instruction is only applied if HGTransmitMode , or HTTransmitMode has selected one of the MULTI_BURST_MODE, or CONTINUOUS_BURST mode selections. Use HGRun , HGStart , and HTRun to start the transmission of the bursts.

HGClearGroup

Description	Ungroups a number of ports that were previously grouped together with the HGSetGroup command.
Syntax	int HGClearGroup(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Since there can only be one group defined at a time, HGClearGroup needs no arguments.

HGClearPort

Description	Clears internal counters from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGClearPort(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command is used on SmartCards. For Passive Hub cards, use the HGClear command.

HGCollision

Description	Determines the collision mode and count for the 100 Mbits Fast SmartCards.
Syntax	int HGCollision(CollisionStructure* CStruct)
Parameters	<i>CStruct</i> CollisionStructure* Holds information pertaining to the collision mode (off, on), and count.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	See the definition of CollisionStructure below. The offset and length fields are not used for 100 Mbits SmartCard.

CollisionStructure

Parameter	Description
unsigned Offset	Specifies the offset, in bits, starting from the first bit of the preamble where the collision is to take place. This value is only used when the Collision Mode is COLLISION_ADJ, CORP_A or CORP_B. It is ignored when the Collision Mode is COLLISION_LONG. The Offset value entered here also pertains to the collisions produced on the SmartBits when it is attached to the ET-1000. <i>Range: 0 to 65535 (0x0000 to 0xFFFF).</i>
unsigned uration	Specifies the duration in bits that the collision is to be asserted. This value is used only when the Collision Mode is COLLISION_ADJ, CORP_A or CORP_B. It is ignored when the Collision Mode is COLLISION_LONG. Note that the Duration value entered here also pertains to the collisions produced on the SmartBits when it is attached to the ET-1000. <i>Range: 1 to 65535 (0x0000 to 0xFFFF). A duration of 0 is invalid.</i>

Parameter	Description										
int Count	Specifies the number of consecutive collisions to produce (one in each packet) before the collision goes inactive. A count of 0 essentially disables the collision counting mechanism, thus producing continuous collisions of the specified type. Note that the Duration value entered here also pertains to the collisions produced on the SmartBits when it is attached to the ET-1000. <i>Range: 0 to 1024.</i>										
int Mode	Specifies the collision mode. <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;">COLLISION_OFF</td> <td>Collision Off</td> </tr> <tr> <td style="padding-left: 20px;">COLLISION_LONG</td> <td>Long Collision</td> </tr> <tr> <td style="padding-left: 20px;">COLLISION_ADJ</td> <td>Adjustable Collision (on transmission)</td> </tr> <tr> <td style="padding-left: 20px;">CORP_A</td> <td>Collision on receive packet, Port A</td> </tr> <tr> <td style="padding-left: 20px;">CORP_B</td> <td>Collision on receive packet, Port B</td> </tr> </table>	COLLISION_OFF	Collision Off	COLLISION_LONG	Long Collision	COLLISION_ADJ	Adjustable Collision (on transmission)	CORP_A	Collision on receive packet, Port A	CORP_B	Collision on receive packet, Port B
COLLISION_OFF	Collision Off										
COLLISION_LONG	Long Collision										
COLLISION_ADJ	Adjustable Collision (on transmission)										
CORP_A	Collision on receive packet, Port A										
CORP_B	Collision on receive packet, Port B										

HGCollisionBackoffAggressiveness

Description	Determines the wait factor for backing off from multiple collisions only on SmartCards in a previously selected group.
Syntax	int HGCollisionBackoffAggressiveness(unsigned int uiAggressiveness)
Parameters	<i>uiAggressiveness</i> unsigned int Set the backoff factor. The length of time actually delayed follows as powers of two using this factor.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGCRC

Description	Create packets with CRC errors on the previously selected group. This function is valid for SmartCards only.
Syntax	int HGCRC(int iMode)
Parameters	<i>iMode</i> int Set the error facility on or off. Valid flags: ET_ON and ET_OFF
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGDataLength

Description	This command is used to specify the length of the data field in the packets being transmitted by the SmartBits ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. Applies only to SmartCards. A random packet size can also be selected.
Syntax	int HGDataLength(int iLength)
Parameters	<i>iLength</i> int Specifies the length of the packets that are to be transmitted on the addressed port. The length is specified in bytes, and it includes everything between the preamble and the CRC. The actual transmitted packet will be extended four bytes for the CRC. Length can range from 1 to 8191. A Length of 0 will cause random packet sizes to be transmitted.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGDribble

Description	Create dribble bit errors on the previously selected group. This function is valid for SmartCards only.
Syntax	int HGDribble(int iBits)
Parameters	<i>iBits</i> int Sets the number of dribble bits to transmit. Valid range is 0 to 7. Setting this value to 0 disables generation of packets with dribble bit errors.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGDuplexMode

Description	Indicates whether to set full duplex or half duplex mode for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGDuplexMode(int iMode)
Parameters	<i>iMode</i> int Sets the Duplex mode where iMode should be one of the following: FULLDUPLEX_MODE Full duplex mode on HALFDUPLEX_MODE Half duplex mode on
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGFillPattern

Description	Specifies the data pattern that is to be transmitted from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. Any VFD data will overwrite this pattern.
Syntax	int HGFillPattern(int iSize, int* piData)
Parameters	<p><i>iSize</i> int Identifies the size, in bytes, of the fill pattern contained in the Data array. Size may range from 60 to 2044. A value of 0 (zero) will cause a random data pattern to be generated.</p> <p><i>piData</i> int* Points to the array which contains the data pattern to be transmitted. A value of NULL will cause a random data pattern to be generated.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See Appendix B for error codes.
Comments	None

HGGap

Description	Specifies the inter-packet gap that is to be transmitted from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. Also allows random gaps to be transmitted.
Syntax	int HGGap(long IPeriod)
Parameters	<p><i>IPeriod</i> long On 10Mbit cards, this value equals the number of tenths of microseconds between transmitted packets.</p> <p>On 100Mbit cards, this value equals the number of tens of nanoseconds between transmitted packets.</p> <p>In either case, IPeriod may range from 10 (=1us) to 1,600,000. A value of 0 (long) will cause a random gap to be generated. For example, if IPeriod = 96, for 10Mbit cards, the Gap will be $96 \times 0.1\mu s = 9.6\mu s$, and for 100Mbit cards, the Gap will be $96 \times 10ns = 960ns$. In both cases, the cards get the minimum legal interpacket gap.</p> <p>For TokenRing cards at 4Mbit, the minimum legal "Gap" is 250ns, and for TokenRing cards at 16Mbits, the minimum legal "Gap" is 65ns.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See Appendix B for error codes.
Comments	None

HGGapAndScale

Description	Specifies the inter-packet gap that is to be transmitted on the addressed port. Also allows random gaps to be transmitted. Applies only to SmartCards.
Syntax	int HGGapAndScale(long IPeriod, int iScale)
Parameters	<p><i>IPeriod</i> long Identifies the number of “scaled” units to be between transmitted packets. A value of 0 (long) will cause a random gap to be generated.</p> <p><i>iScale</i> int Determines the scale that the <i>IPeriod</i> parameter based on: NANO_SCALE = nanoseconds scale MICRO_SCALE = microseconds scale MILLI_SCALE = milliseconds scale</p> <p>Possible values range from 1 to 1,600,000,000 nanoseconds for NANO_SCALE; from 1 to 1,600,000 microseconds for MICRO_SCALE; and from 1 to 1600 milliseconds for MILLI_SCALE. Values outside this range return an error code.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	None

HGGetCounters

Description	Retrieves counters from all ports in the group defined by the previous HGSetGroup/HGAddtoGroup command. This information is placed into the HTCCountStructures pointed to in the input argument.
Syntax	int HGGetCounters(HTCCountStructure* phtCountStruct)
Parameters	<i>phtCountStruct</i> HTCCountStructure* A pointer to the first element of an array of counter structures in which count information is to be placed. See the definition of HTCCountStructure below.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	It is assumed that the calling function has declared the HTCCountStructure array and reserved sufficient memory for it.

HGGetEnhancedCounters

Description	Retrieves standard counters and card related counters from all ports in the group defined by the previous HGSetGroup/HGAddtoGroup commands. This information is placed into the EnhancedCountersStructure pointed to in the input argument. Applies to SmartCards and TokenRing SmartCard. NOTE: This command may not be used with Gigabit Ethernet SmartCards and SmartModules.
Syntax	int HGGetEnhancedCounters(EnhancedCountersStructure* pEnCounter)
Parameters	<i>pEnCounter</i> EnhancedCountersStructure* A pointer to the first element of an array of counter structures in which count information is to be placed. See the description of EnhancedCountersStructure below.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	It is assumed that the calling function has declared the EnhancedCountersStructure array and reserved sufficient memory for it.

EnhancedCountersStructure

Parameter	Description
int iMode	Counter mode control. 0 Set to Count 1 Set to Rate
int iPortType	Card type is returned in this member variable. CT_ACTIVE 10Mb Ethernet CT_FASTX 10/100Mb Ethernet CT_TOKENRING 4/16Mb TokenRing CT_VG VG/AnyLan
unsigned long ulMask1	Bit mask for the Standard counters. The Standard counter type can be any one, (or a combination calculated by performing a bit-wise "or") of the applicable constants below: SMB_STD_TXFRAMES Transmitted Packets SMB_STD_TXBYTES Transmitted Bytes SMB_STD_TXTRIGGER Transmitted Trigger Packets SMB_STD_RXFRAMES Received Packets SMB_STD_RXBYTES Received Bytes SMB_STD_RXTRIGGER Received Trigger Packets SMB_STD_ERR_CRC Checksum Packets SMB_STD_ERR_ALIGN Alignment Packets SMB_STD_ERR_UNDERSIZE Undersized Packets SMB_STD_ERR_OVERSIZE Oversized Packets SMB_STD_ERR_COLLISION Collision Packets Get a combination of the above by "OR-ing" together criteria from the above list.

Example for ulMask1

```
EnhancedCountersStructure ECSTx;
int iErr = 0;
memset(&ECSTx, 0, sizeof(ECSTx));
ECSTx.ulMask2 = L3_ARP_REQ + L3_ARP_REPLIES;
iErr = HTGetEnhancedCounters(&ECSTx, TxHub, TxSlot, TxPort);
printf (msg, "ECSTx Arp Requests: %u\n", ECSTx.ulData[39]);
printf (msg, "ECSTx Arp Replies: %u\n", ECSTx.ulData[41]);
```

Parameter	Description
unsigned long ulMask2	Bit mask for the Additional counters on some SmartCards and SmartModules. The Additional counter type can be any of the applicable constants below (or a combination calculated by performing a bit-wise "OR"): Token Ring SmartCard. The following are recognized in ulMask2: TR_MASK Allowable possible bits. TR_LATENCY Latency time in 100ns counts. TR_TOKEN_RT Rotation time in microseconds. Counters indicated by TR_MAC are derived from Ring Error Monitor MAC frames, others are from direct counts. Consult the TR architectural specification for the definition of these counts. TR_RXMAC Received MAC frames. Mac frames are used to manage a ring. TR_RXABORTFRAMES Abort Frames. These frames end with an "Abort Delimiter" rather than the normal "End Delimiter." These are frames that the transmitter stopped sending before they were complete. TR_LINEERRORS Line errors counter. Line errors occur when the line ceases to have signal for a designated length of

	<p>time. Typically this is caused by an unplugged wire.</p> <p>TR_BURSTERRORS Burst errors counter. Burst Errors are when the line is disconnected for a short time, typically less than 5 bit times.</p> <p>TR_BADTOKEN Corrupted tokens. Bad Tokens are when there is garbage instead of tokens (which look like small frames).</p> <p>TR_PURGEEVENTS Purge MAC frames detected. The presence of "Purge" MAC frames occurs just before the ring starts working normally.</p> <p>TR_BEACONEVENTS Beacon MAC frames detected. Beacons are MAC frames used to determine if the ring is complete. Stations send them if they can't establish a ring.</p> <p>TR_CLAIMEVENTS Claim MAC frames detected. Claims are MAC frames used to let stations bid to throw and monitor the token.</p> <p>TR_INSERTIONS Request initializations. Request Initialization frames are MAC frames sent as a station joins the ring. They can be used to indicate how often stations join the ring.</p> <p>The MAC type error counts below are taken from "Ring Error Monitor" reporting frames. Stations keep track of errors Internally. Periodically, (or when the counters overflow), they report the errors to the "Ring Error Monitor." For your convenience, SmartLib tracks these errors. This information, however, will not be as complete as that from a program such as "LAN Manager."</p> <p>For definitions of the errors below, see the <i>Architectural Reference</i> or standards documents.</p> <p>TR_MAC_LINEERRORS Isolating line error.</p> <p>TR_MAC_INTERNALERRORS Internal error.</p> <p>TR_MAC_BURSTERRORS Burst errors.</p> <p>TR_MAC_ACERRORS AMP detects circulating frame.</p> <p>TR_MAC_ABORTTX Abort delimiter detected.</p> <p>TR_MAC_LOSTFRAME Incompletely stripped frame.</p> <p>TR_MAC_RXCONGESTED Receiver congestion.</p> <p>TR_MAC_FRAMECOPIED Possible duplicate address.</p> <p>TR_MAC_FREQUENCYERROR Excessive jitter detected.</p> <p>TR_MAC_TOKENERROR Circulating frames.</p> <p>SMB_VG_MASK Allowable possible bits.</p> <p>VG SmartCard. The following are recognized in ulMask2:</p> <p>SMB_VG_INV_PKTMARK Invalid packet marker errors.</p> <p>SMB_VG_ERR_PKT Errored packets received.</p> <p>SMB_VG_TRANSTRAIN_PKT Transition into training.</p> <p>SMB_VG_PRIO_PROM_PKT Priority promoted packets received or transmitted.</p> <p>L3_MASK Allowable possible bits.</p> <p>Layer 3 SmartCards. The following are recognized in ulMask2:</p> <p>L3_FRAMEERROR Framing errors. Framing Errors, caused by dribbling, occur when the total number of bits received by the card is not a multiple of 8. On a 10 Mbps card, 1 to 7 additional bits are</p>
--	--

	<p>L3_TX_RETRIES</p> <p>L3_TX_EXCESSIVE</p> <p>L3_TX_LATE</p> <p>L3_RX_TAGS</p> <p>L3_TX_STACK</p> <p>L3_RX_STACK</p> <p>L3_ARP_REQ</p> <p>L3_ARP_SEND</p> <p>L3_ARP_REPLIES</p> <p>L3_PINGREP_SENT</p> <p>L3_PINGREQ_SENT</p> <p>L3_PINGREQ_RECV</p>	<p>possible. On a 100 Mbps card, the error is off by 4 bits.</p> <p>Number of transmit collisions/retries.</p> <p>Number of times a frame needed more than 16 retries. (Available only for L3-6705 and L3-6710.)</p> <p>Number of collisions that occurred more than 64 bytes into a frame. (Available only for L3-6705 and L3-6710.)</p> <p>Number of number of received frames that have "signature" fields.</p> <p>Number of frames transmitted from the SmartCard's local stack.</p> <p>Number of Number of frames received by the SmartCard's local stack.</p> <p>Number of ARP request frames originating on the SmartCard.</p> <p>Number of ARP reply frames originating on the SmartCard.</p> <p>Number of ARP request frames received by the SmartCard.</p> <p>Number of ICMP Ping reply frames sent by the SmartCard.</p> <p>Number of ICMP Ping request frames sent by the SmartCard.</p> <p>Number of ICMP Ping request frames received by the SmartCard.</p>
unsigned long ulData[64]	Array of counters returned. ulMask1 and ulMask2 are bit masks that identify the 64 possible counters, with bit 0 of ulMask1 corresponding to ulData[0], bit 1 of ulMask1 corresponding to ulData[1], bit 0 of ulMask2 corresponding to ulData[32] and so on.	

HGGetGroupCount

Description	Returns the number of ports currently configured in the group.
Syntax	int HGGetGroupCount(void)
Parameters	None
Return Value	Returns the number of ports currently configured in the group. The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	

HGGetLEDs

Description	Determine the state of the LEDs on ports in the currently defined group.
Syntax	int HGGetLEDs(int* piLEDs)
Parameters	piLEDs int* a pointer to an integer array of at least the number of cards in the group size that receives the LED states of all SmartCards in the current group.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Behavior of this function is undefined if the port group contains passive cards.

HGIsPortInGroup

Description	Returns whether the specified port is currently configured in the group.
Syntax	int HGIsPortInGroup(int iPortId)
Parameters	<i>iPortId</i> int the counting ordinal ID of the port in the test bay whose inclusion in the group is to be checked.
Return Value	Returns a positive (non-zero) number if TRUE, zero if FALSE. The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGIsHubSlotPortInGroup

Description	Returns whether the specified port is currently configured in the group.
Syntax	int HGIsHubSlotPortInGroup(int Hub, int Slot, int Port)
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. <i>See Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	Returns a positive (non-zero) number if TRUE, zero if FALSE. The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGMultiBurstCount

Description	Sets up the number of bursts for transmitting out a SmartCard while in MULTI_BURST_MODE.
Syntax	int HGMultiBurstCount(long lVal)
Parameters	<i>lVal</i> long Specifies the burst count. <i>Range:</i> 1 to 16,777,215.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This instruction is only applied if HGTransmitMode , or HTTransmitMode has selected MULTI_BURST_MODE. Use HGRun , HGStart , and HTRun to start the transmission of the bursts.

HGRemoveFromGroup

Description	Along with HGSetGroup, this command can be used to remove individual hub/slot/port designations from a currently configured group.
Syntax	int HGRemoveFromGroup (int iHub, int iSlot, int iPort)
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HGRemovePortIdFromGroup

Description	Can be used to remove individual hub/slot/port designations from a currently configured group that has been set up using HGSetGroup.
Syntax	int HGRemovePortIdFromGroup (int iPortId)
Parameters	<i>iPortId</i> int Identifies the port which is to be removed from the currently configured group. The value used for the iPortId is determined from the ordinal counting number of existing ports in the test bay.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	<p>The first hub in the daisy chain from the control section would contain the first set of ports to be identified. The port in the left-most (lowest numbered) slot in the first hub is identified as $iPortId = 1$, the next port in the sequence going left to right across the slots, would be identified as $iPortId = 2$, and so on until all existing ports in the first hub have been identified. Any empty slots are skipped over for the purposes of assigning PortId numbers. The next hub in the daisy chain connection (at the back of the test bay) would then continue with the next counting number as the iPortId identifier.</p> <p>Example 1: Assume you have a 4-hub test bay with 20 ports in each hub. Then the ports in the first hub are identified left to right as ports 1 through 20. The second hub ports are identified left to right as ports 21 through 40. The third hub ports are identified left to right as ports 41 through 60. And the fourth hub ports are identified left to right as ports 61 through 80.</p> <p>Example 2: Assume you have a four hub test bay with 7 ports in the first hub, 4 ports in the second hub, no ports in the third hub and 3 ports in the fourth hub. The first hub ports are identified left to right as ports 1 through 7. The second hub ports are identified left to right as ports 8 through 11. The third hub is skipped over as any other empty slots are and the counting continues at the next port, which happens to be in the fourth hub. The ports in the fourth hub are then identified left-to-right as ports 12 through 15.</p>

HGResetPort

Description	Resets the SmartCards defined in the current group to a default condition with all errors off.
Syntax	int HGResetPort(int iResetType)
Parameters	<p><i>iResetType</i> int Identifies the run mode of the board. Legal modes can be conveyed using the following constants:</p> <p>RESET_FULL Reset all card parameters including hardware interface parameters (e.g. Token Ring Speed)</p> <p>RESET_PARTIAL Reset all card parameters except hardware interface parameters (e.g. Token Ring Speed)</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command is not implemented on the ATM and WAN (FR) SmartCards at this time.

HGRun

Description	<p>Sets up the run state for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.</p> <p>The port can be set up to transmit a series of packets ("RUN" state), transmit a single packet ("STEP" state) or stop transmission altogether ("STOP" state).</p> <p>If the Burst mode has been set up to transmit a burst of packets (using the HTTransmit command), then transitioning from "STOP" to "RUN" will cause the specified number of packets to be transmitted.</p> <p>This command works with HTSeparateHubCommands. If no setting is specified, the default used for HGRun is HUB_DEFAULT_ACTION.</p>
Syntax	int HGRun(int iMode)
Parameters	<p><i>iMode</i> int Identifies the run mode of the board. Legal modes can be conveyed using the following constants:</p> <p>HTRUN</p> <p>HTRUN_VALUE Transmit continuously or send a burst of packets. **Use HTRun_Value for Visual Basic.**</p> <p>HTSTEP Transmit a single packet.</p> <p>HTSTOP Halt transmission of packets altogether.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>Because Visual Basic does not distinguish by case, this value has been put in the ETSMBAPI.TXT file:</p> <p>HTRUN_VALUE Transmit continuously or send a burst of packets.</p>

HGSelectTransmit

Description	Enables the Port B transmission of the SmartBits to be transmitted to the ports in the currently defined group. Transmission mode is determined by <i>iMode</i> .
Syntax	int HGSelectTransmit(int iMode)
Parameters	<p><i>iMode</i> int Determines the function of the Port:</p> <p>HTTRANSMIT_OFF Transmitter is turned off</p> <p>HTTRANSMIT_STD Transmitter transmits standard packets</p> <p>HTTRANSMIT_COL Transmitter transmits collision packets</p> <p>All other values are invalid and have no effect on the SmartBits.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This function assumes that at least one SmartBits is attached to the SmartBits. It will be ignored by the SmartBits if there is not an SmartBits present.

HGSetGroup

Description	Groups a number of SmartBits ports. These ports may then be manipulated as a group using the any of the SmartLib HG commands.
Syntax	int HGSetGroup(char* pszPortIdGroup)
Parameters	<p><i>pszPortIdGroup</i> char* A NULL terminated ASCII character string with a maximum of 512 characters. This string defines which ports are members of the active group.</p> <p>Although a port is usually specified by identifying the iHub, iSlot, and iPort, group members are identified by a single number. This number is the actual sequence number of the port - with numbers starting at the Master controller.</p> <p>The pszPortIdGroup numbers:</p> <ul style="list-style-type: none"> * Start at 1 (as opposed to 0). * Do not count blank slots as part of the sequence. * Do not account for the hub number. <p>Example: If you had four different hubs with one card each, you could include them all in the group with these values: "1,2,3,4"</p> <p>Ports may be separated by commas and/or spaces. Any number of commas or blank spaces may be inserted between the port numbers, as long as the overall length of the string doesn't exceed 512.</p> <p>Dashes may also be used to identify the group. For example: "1-100, 105, 256" groups the first one hundred ports as well as the hundred and fifth, and the two hundred and fifty-sixth port.</p> <p>You can group ports in ascending or descending order so that "4 - 1" is a valid value.</p> <p>Port numbers are assigned from left to right, top to bottom, first link to last link.</p> <p>To clear an old group selection, use HGClearGroup. You can also pass NULL as the PortIdGroup.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>

Comments	<p>Only one group can exist at a time. All HG commands will act upon the last PortIdGroup defined by HGSetGroup(PortIdGroup). Groups may be defined and redefined at any time. See also HGAddtoGroup.</p> <p>The first hub in the daisy chain from the control section would contain the first set of ports to be identified. The port in the left-most (lowest numbered) slot in the first hub is identified as iPortId = 1, the next port in the sequence going left to right across the slots, would be identified as iPortId = 2, and so on until all existing ports in the first hub have been identified. Any empty slots are skipped over for the purposes of assigning PortId numbers. The next hub in the stack would then continue with the next counting number as the iPortId identifier.</p>
-----------------	--

HGSetGroupType

Description	Reserves a group of ports by card types within a SmartBits configuration. These ports may then be manipulated simultaneously with one another (as a group) using the any of the HG commands defined herein.
Syntax	int HGSetGroupType(int Index, int* pPortIdList)
Parameters	<p><i>Index</i> int Size of card type array. The default setting is CT_MAX_CARD_TYPE. A value of -1 will select all types of cards, a value of 0 will clear the group selection.</p> <p><i>pPortIdList</i> int* An array of integers which describes the ports that are to be grouped. pPortIdList[0] is designates CT_ACTIVE (10MB Ethernet) card types to be included in the group. PPortIdList[1] is for CT_PASSIVE card types, pPortIdList[2] is for CT_FASTX card types, and so on for each of the CT_xxx card types.</p> <p>For each value of pPortIdList[]:</p> <p style="padding-left: 40px;">0 means do not select this card type,</p> <p style="padding-left: 40px;">1 means to include this card type in the group.</p> <p><i>For example:</i></p> <p><i>Index = 8, and {0, 0, 1, 1, 0, 0, 0, 1} will select all the FAST, TOKENRING, and GIGABIT cards.</i></p> <p>To clear an old group selection, pass 0 in the Index.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>Only one group can exist at any time for the HG commands.</p> <p>Groups can cross hub boundaries.</p> <p>Groups may be defined and redefined at any time.</p> <p>All HG commands will act upon the last PortIdList defined by HGSetGroupType(Index, PortIdList). This command can be used to reset a group previously set by HGSetGroup command.</p>

HGSetSpeed

Description	Sets selected speed for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. The speed selected must be appropriate to the addressed SmartCard type.
Syntax	int HGSetSpeed(int iSpeed)
Parameters	<p><i>iSpeed</i> int Determines the speed of the Port:</p> <p>SPEED_10MHZ Sets a 10MB capable SmartCard to a 10 MHZ Signaling rate</p> <p>SPEED_100MHZ Sets a 100MB capable SmartCard to a 100 MHZ Signaling rate</p> <p>SPEED_4MHZ Sets a 4MB capable SmartCard to a 4 MHZ Signaling rate</p> <p>SPEED_16MHZ Sets a 16MB capable SmartCard to a 16 MHZ Signaling rate</p> <p>SPEED_155MHZ Sets a 155MB capable SmartCard to a 155 MHZ Signaling rate</p> <p>SPEED_25MHZ Sets a 25MB capable SmartCard to a 25 MHZ Signaling rate</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGSetTokenRingAdvancedControl

Description	Generates specialized frames for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. This command only works for the TokenRing SmartCard.
Syntax	int HGSetTokenRingAdvancedControl(TokenRingAdvancedStructure *pTRAdvancedStructure)
Parameters	<i>pTRAdvancedStructure</i> TokenRingAdvancedStructure* Points to a TokenRingAdvancedStructure (see below), which contains all the information required to transmit special control frames.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command will cause ring operation to fail if not used with full knowledge of the <i>Token Ring Architectural Specification</i> .

TokenRingAdvancedStructure

Parameter	Description
int UseHoldingGap	Token holding gap control. 1 Activate advanced gap control. 0 Do not issue advanced gap control.
int GapValue	Time between frames when the token is not released between frames. <i>Range:</i> 1 to 1,600,000 (number of 100 nanosecond periods between frames). Default value: 1.

Parameter	Description
int GapScale	Scale value. NANO_SCALE Scale in nanoseconds MICRO_SCALE Scale in microseconds MILLI_SCALE Scale in milliseconds
int UseIntermediate FrameBits	Sets the Intermediate frame bit in the EDEL field of the frame. This bit is defined in the Token Ring Specification to indicate that another frame is to follow immediately, with no token being released between the frames. (See the Token Ring Architectural Specification.) 1 Set Intermediate frame 0 Clear Intermediate frame
int UseAC	Activates a user-specified Access Control field in transmitted frames. 1 Set AC from ACdata field 0 Set AC from captured token
int ACdata	Access Control byte value. Consult the Token Ring Architectural Specification for information on the bit fields in this byte. This byte is used to distinguish between tokens and frames and to operate the Token Priority Protocol. Setting bits in this byte incorrectly will probably cause ring errors.
int AdvancedControl1	Advanced control byte 1. This byte gives the user control over how the card connects to the ring on startup and how it responds to ring errors. Bit 3-2: Controls connection on startup. 0 No effect (previous settings in NVRAM are used). 1 Connects to the ring on startup (default). 2 Stays off the ring on startup. 3 Stays off the ring on startup and allows bit 1 to control the connection. Bit 1: Connection control 0 Deinserted 1 Inserted Bit 0: 'Halt on Error' Stops card from transmitting when it receives a Beacon, Claim or Purge frame. 0 Inactive 1 Active
int AdvancedControl2	Advanced control byte 2. Bit 4: Internal Loopback 0 Off 1 On Bit 3: Test Mode. This mode is used to simulate an Active Monitor when running as a Station so that the card can be used standalone to test passive Token Ring components. 0 Off 1 On
unsigned long AReserved1	Reserved.
unsigned long AReserved2	Reserved.

HGSetTokenRingErrors

Description	Generates error frame traffic simultaneously for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. This command applies only to the TokenRing SmartCard.												
Syntax	int HGSetTokenRingErrors(int ErrorTrafficRatio, int iTRErrors)												
Parameters	<p><i>ErrorTrafficRatio</i> int Specifies the error traffic ratio in tenths of percent. <i>Range:</i> 0 to 1000. A value of 0 turns off error generation.</p> <p><i>iTRErrors</i> int Specifies the type of frame errors to generate. Value can be a combined OR of the following defines:</p> <table border="0"> <tr> <td>TR_ERR_FCS</td> <td>FCS errors</td> </tr> <tr> <td>TR_ERR_FRAME_COPY</td> <td>Frame copy errors</td> </tr> <tr> <td>TR_ERR_FRAME_BIT</td> <td>Frame Bit errors</td> </tr> <tr> <td>TR_ERR_FRAME_FS</td> <td>FS Frame errors</td> </tr> <tr> <td>TR_ERR_ABORT_DELIMITER</td> <td>Abort delimiter errors</td> </tr> <tr> <td>TR_ERR_BURST</td> <td>Burst errors</td> </tr> </table>	TR_ERR_FCS	FCS errors	TR_ERR_FRAME_COPY	Frame copy errors	TR_ERR_FRAME_BIT	Frame Bit errors	TR_ERR_FRAME_FS	FS Frame errors	TR_ERR_ABORT_DELIMITER	Abort delimiter errors	TR_ERR_BURST	Burst errors
TR_ERR_FCS	FCS errors												
TR_ERR_FRAME_COPY	Frame copy errors												
TR_ERR_FRAME_BIT	Frame Bit errors												
TR_ERR_FRAME_FS	FS Frame errors												
TR_ERR_ABORT_DELIMITER	Abort delimiter errors												
TR_ERR_BURST	Burst errors												
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>												
Comments	The number in the ratio is nominally in tenths of a percent. However, as it is rationalized to a count the precision will be poor at large percentage values.												

HGSetTokenRingLLC

Description	Configures an LLC frame simultaneously for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. This command applies only to the TokenRing SmartCard.
Syntax	int HGSetTokenRingLLC(TokenRingLLCStructure *pTRLStructure)
Parameters	<i>pTRLStructure</i> TokenRingLLCStructure* Points to a TokenRingLLCStructure (see below), which contains all the information required to preform LLC Type 1 frames.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	TokenRing MAC header also has to be defined for this command to take effect.

TokenRingLLCStructure

Parameter	Description
int UseLLC	Logical Link Control (LLC). 0 No LLC added to MAC frame header. 1 Add LLC to the MAC frame header.
int DSAP	Destination Service Access Point. Range: 0 to 255 (0x00 to 0xFF).
int SSAP	Source Service Access Point. Range: 0 to 255 (0x00 to 0xFF).
int LLCCommand	Sets the type of LLC field to be added to the frame header. 0 TEST frame set to 'Poll' 1 SNAP frame (used to encapsulate an Ethernet frame from the 'type' field)

HGSetTokenRingMAC

Description	Configures a TokenRing MAC header simultaneously for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. This command applies only to the TokenRing SmartCard.
Syntax	int HGSetTokenRingMAC(TokenRingMACStructure *pTRMStructure)
Parameters	<i>pTRMStructure</i> TokenRingMACStructure* Points to a TokenRingMACStructure (see page Error! Bookmark not defined.), which defines a preformed MAC header.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

TokenRingMACStructure

Parameter	Description																
int UseMAC	MAC header control. The MAC header consists of AC and FC bytes, followed by MAC destination and source addresses, followed by optional LLC control, followed by optional SourceRouteAddress information. AC and FC are always prepended to frame data. <table style="margin-left: 40px;"> <tr> <td>0</td> <td>No MAC header prepended to frame data.</td> </tr> <tr> <td>1</td> <td>Prepend a MAC header to the frame data.</td> </tr> </table>	0	No MAC header prepended to frame data.	1	Prepend a MAC header to the frame data.												
0	No MAC header prepended to frame data.																
1	Prepend a MAC header to the frame data.																
int Stations	Reserved. Must be set to 1.																
int MACSrc[6]	Source MAC Address.																
int MACDest[6]	Destination MAC Address.																
int FramesPerToken	Source MAC Address.																
int FrameControl	This is the value of the Frame Control byte put on the front of each frame. This byte is independent of the fill pattern and any pre-formed header may be overwritten by a VFD field. This byte is defined fully in the Token Ring Architectural Specification and should not be altered from the default value of 0x40 (TRFC_DEFAULT) without knowledge of the consequences. Several other values are defined in the header file: <table style="margin-left: 40px;"> <tr> <td>TRFC_DEFAULT</td> <td>Standard frame</td> </tr> <tr> <td>TRFC_PCF_BEACON</td> <td>Beacon</td> </tr> <tr> <td>TRFC_PCF_CLAIMTOKEN</td> <td>Claim Token</td> </tr> <tr> <td>TRFC_PCF_RINGPURGE</td> <td>Ring Purge</td> </tr> <tr> <td>TRFC_PCF_AMP</td> <td>Active Monitor Present</td> </tr> <tr> <td>TRFC_PCF_SMP</td> <td>Standby Monitor Present</td> </tr> <tr> <td>TRFC_PCF_DAT</td> <td>Duplicate Address Test</td> </tr> <tr> <td>TRFC_PCF_RRS</td> <td>Remove Ring Station</td> </tr> </table>	TRFC_DEFAULT	Standard frame	TRFC_PCF_BEACON	Beacon	TRFC_PCF_CLAIMTOKEN	Claim Token	TRFC_PCF_RINGPURGE	Ring Purge	TRFC_PCF_AMP	Active Monitor Present	TRFC_PCF_SMP	Standby Monitor Present	TRFC_PCF_DAT	Duplicate Address Test	TRFC_PCF_RRS	Remove Ring Station
TRFC_DEFAULT	Standard frame																
TRFC_PCF_BEACON	Beacon																
TRFC_PCF_CLAIMTOKEN	Claim Token																
TRFC_PCF_RINGPURGE	Ring Purge																
TRFC_PCF_AMP	Active Monitor Present																
TRFC_PCF_SMP	Standby Monitor Present																
TRFC_PCF_DAT	Duplicate Address Test																
TRFC_PCF_RRS	Remove Ring Station																

HGSetTokenRingProperty

Description	Simultaneously configures ring operation characteristics for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. This command only works for TokenRing SmartCard.
Syntax	int HGSetTokenRingProperty(TokenRingPropertyStructure *pTRPStructure)
Parameters	<i>pTRPStructure</i> TokenRingPropertyStructure* Points to a TokenRingPropertyStructure (see below), which contains all the information required to configure ring operation.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	This command defines card properties that are retained in non-volatile storage. These parameters should not be altered on a live ring as they will probably cause ring malfunction (usually Beaconsing by other stations, which might cause them to close down pending a hard reset).

TokenRingPropertyStructure

Parameter	Description
int SpeedSetting	Ring speed. TR_SPEED_4MBITS 4 Mbits/Sec TR_SPEED_16MBITS 16 Mbits/Sec
int EarlyTokenRelease	Allows a station to transmit a token immediately after a frame was sent. This feature only applies to a ring running at 16 Mbps. TR_TOKEN_DEFAULT Do not allow. TR_TOKEN_EARLY_RELEASE Allow.
int DuplexMode	Half duplex or full duplex. TR_DUPLEX_HALF TKP Half duplex TR_DUPLEX_FULL TXI Full duplex
int DeviceOrMAUMode	Configures the TokenRing SmartCard to be a port or a station. TR_MODE_MAU Port TR_MODE_DEVICE Station

HGSetTokenRingSrcRouteAddr

Description	Configures a Source Route Address(SRA) simultaneously for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. This command applies only to the TokenRing SmartCard.
Syntax	int HGSetTokenRingSrcRouteAddr(int UseSRA, int *piData)
Parameters	<p><i>UseSRA</i> int specifies whether an SRA field will be included in a TokenRing frame.</p> <p> 0 No SRA defined</p> <p> 1 Use SRA defined in piData parameter.</p> <p><i>piData</i> int * Points to an array of int which contains the Source Route Address information. The maximum length of this array is 32 and the length information is encoded in the lower 5 bits of the first byte of SRA.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	This field is part of a pre-formed header and so the MAC header has to be active for it to be active. The data in this field will be parsed by the card to determine the size of the source routing field to use and the maximum frame size to transmit. (See the <i>Token Ring Architectural Reference</i> for details of how to code this field.)

HGSetVGProperty

Description	Configures VG SmartCards operating characteristics simultaneously for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGSetVGProperty(VGCardPropertyStructure *pVGPStructure)
Parameters	<i>pVGPStructure</i> VGCardPropertyStructure* Points to a VGCardPropertyStructure (see below), which contains all the information required to configure VG Cards.
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	None

VGCardPropertyStructure

Parameter	Description
int EndOrMasterNode	Allows a VG SmartCard to be configured as an End node or a Master node. VG_CFG_END_NODE End Node VG_CFG_MASTER Master Node
int PriorityPromotion	Priority promotion. VG_CFG_NO_PRIO_PROMO No promotion VG_CFG_PRIORITY_PROMO Yes
int EtherNetOrTokenRing	Configures the VG SmartCard to be operated in Ethernet or in TokenRing. VG_CFG_ETHERNET Ethernet CG_CFG_TOKENRING TokenRing

HGStart

Description	Simultaneously starts transmission of packets from all ports associated with the PortIdGroup defined by previous HGSetGroup(PortIdGroup) command.
Syntax	int HGStart(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command works in conjunction with HTSeparateHubCommands. If no setting is specified, the default used for HGStart is HUB_DEFAULT_ACTION.

HGStep

Description	Simultaneously causes the transmission of a single packet or burst from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGStep(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command works in conjunction with HTSeparateHubCommands. If no setting is specified, the default used for HGStep is HUB_DEFAULT_ACTION.

HGStop

Description	Simultaneously halts the transmission of packets from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGStop(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This command works in conjunction with HTSeparateHubCommands. If no setting is specified, the default used for HGStop is HUB_DEFAULT_ACTION.

HGSymbol

Description	Generates symbol errors for the 100 Mbps SmartCards. The group of ports can be set up to transmit a series of packets which generates invalid wave form data pattern. This command applies only to 100 Mbits SmartCards.
Syntax	int HGSymbol(int Mode)
Parameters	<i>iMode</i> int Identifies the symbol mode of the board. Legal modes can be conveyed using the following constants: SYMBOL_OFF Turn off symbol errors SYMBOL_ON Turn on symbol errors
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGTransmitMode

Description	Indicates how to control the transmission of packets when running for all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGTransmitMode(int iMode)
Parameters	<p><i>iMode</i> int Indicates the mode of operation when transmitting packets according to the following defined values:</p> <p>CONTINUOUS_PACKET_MODE Sets port to transmit single packets continuously.</p> <p>SINGLE_BURST_MODE Sets port to transmit a single burst of packets, and then stop.</p> <p>MULTI_BURST_MODE Sets port to transmit multiple bursts of packets, indicated by the HTMultiBurstCount command, with each burst being separated by the value specified in the HTBurstGap command, and then stop.</p> <p>CONTINUOUS_BURST_MODE Sets port to continuously send bursts of packets with each burst being separated by the value specified in the HxBurstGap command.</p> <p>ECHO_MODE Sets port to transmit a single packet upon receiving a Receive Trigger event.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGTrigger

Description	Sets up the triggering mechanism from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. HTrigger specifies the trigger number (1 or 2), the operational configuration, trigger pattern range, trigger pattern offset, and trigger pattern data.
Syntax	int HGTrigger(int iTrigId, int iConfig, HTriggerStructure* pHTStruct)
Parameters	<p><i>iTrigId</i> int Identifies the trigger source. There are two possible triggers on each SmartCard. They are identified as follows:</p> <p style="margin-left: 40px;">HTTRIGGER_1 Trigger 1</p> <p style="margin-left: 40px;">HTTRIGGER_2 Trigger 2</p> <p><i>iConfig</i> int There are three possible types of configurations for the triggers on the SmartCards:</p> <p style="margin-left: 40px;">HTTRIGGER_OFF Disables the triggering mechanism for TrigId</p> <p style="margin-left: 40px;">HTTRIGGER_ON Enables the triggering mechanism for TrigId</p> <p style="margin-left: 40px;">HTTRIGGER_DEPENDENT Enables the triggering mechanism for TrigId after the other trigger has triggered.</p> <p><i>pHTStruct</i> HTriggerStructure* A structure containing the trigger pattern, offsets and ranges (see below). Note that the maximum range is 6 bytes. Though the range is specified in bytes, the specified number is rounded up to the nearest byte multiple. i.e.; the SmartCards can only trigger on patterns that are a length that is a multiple of 8 bits. The offset ranges from 1 to 12,112 bits (specified in bits).</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>It is possible to misconfigure triggers when using HTTRIGGER_DEPENDENT. A TrigId set to HTTRIGGER_DEPENDENT is to be active after the other TrigId trigger has occurred. So, if trigger 2 is set to be dependent on trigger 1:</p> <p>A properly configured trigger dependent combination would be (the order of the commands does matter):</p> <pre>HGTrigger(HTTRIGGER_1, HTTRIGGER_ON, &TStruct) HGTrigger(HTTRIGGER_2, HTTRIGGER_DEPENDENT, &TStruct)</pre> <p>A misconfigured trigger combination would be:</p> <pre>HGTrigger(HTTRIGGER_1, HTTRIGGER_OFF, &TStruct) HGTrigger(HTTRIGGER_2, HTTRIGGER_DEPENDENT, &TStruct)</pre> <p>Here, trigger 2 will never fire because trigger 1 is off.</p>

HTriggerStructure

Parameter	Description
unsigned Offset	Specifies the number of bit times that pass between the first non-preamble bit and when the trigger word is searched for in the data stream. <i>Range:</i> 0 to 65535 (0x0000 to 0xFFFF), where 0 matches the first bit after the preamble.
int Range	Specifies the size of the trigger word in bytes. <i>Range:</i> 1 to 6.
int Pattern [6]	Array of bytes containing the trigger word. Pattern[0] is the LSByte, Pattern[5] is the MSByte. For triggers 1 and 2, enter the data pattern array in reverse order.

HGVFD

Description	Sends VFD information to all ports in the group defined by the previous HGSetGroup(PortIdGroup) command.						
Syntax	int HGVFD(int VFDId, HTVFDStructure* HStruct)						
Parameters	<p><i>VFDId</i> int Identifies the VFD pattern being addressed. There are a total of three VFD patterns. They are identified as shown below:</p> <table style="margin-left: 40px;"> <tr> <td>HVFD_1</td> <td>VFD Pattern 1</td> </tr> <tr> <td>HVFD_2</td> <td>VFD Pattern 2</td> </tr> <tr> <td>HVFD_3</td> <td>VFD Pattern 3</td> </tr> </table> <p><i>HStruct</i> HTVFDStructure* Structure holds VFD information used with a SmartCard (VFD Configuration, Range, Offset and Pattern) (see below).</p>	HVFD_1	VFD Pattern 1	HVFD_2	VFD Pattern 2	HVFD_3	VFD Pattern 3
HVFD_1	VFD Pattern 1						
HVFD_2	VFD Pattern 2						
HVFD_3	VFD Pattern 3						
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See Appendix B for error codes.						
Comments	See the definition of HTVFDStructure below.						

HTVFDStructure

Parameter	Description														
int Configuration	<p>Determines the capabilities of the VFD being implemented. Select the constant that applies.</p> <p>Configurations specific to VFD1 and VFD2 are:</p> <table style="margin-left: 40px;"> <tr> <td>HVFD_NONE</td> <td>VFD off</td> </tr> <tr> <td>HVFD_RANDOM</td> <td>Random pattern</td> </tr> <tr> <td>HVFD_INCR</td> <td>Incrementing pattern</td> </tr> <tr> <td>HVFD_DECR</td> <td>Decrementing pattern</td> </tr> <tr> <td>HVFD_STATIC</td> <td>Static pattern</td> </tr> </table> <p>Configuration options for VFD3 are:</p> <table style="margin-left: 40px;"> <tr> <td>HVFD_NONE</td> <td>VFD3 off</td> </tr> <tr> <td>HVFD_ENABLED</td> <td>VFD3 on</td> </tr> </table> <p>NOTE: VFD3 operates differently from 1 and 2. It is a large buffer that can be used in segments to create more complex patterns than increment or decrement.</p>	HVFD_NONE	VFD off	HVFD_RANDOM	Random pattern	HVFD_INCR	Incrementing pattern	HVFD_DECR	Decrementing pattern	HVFD_STATIC	Static pattern	HVFD_NONE	VFD3 off	HVFD_ENABLED	VFD3 on
HVFD_NONE	VFD off														
HVFD_RANDOM	Random pattern														
HVFD_INCR	Incrementing pattern														
HVFD_DECR	Decrementing pattern														
HVFD_STATIC	Static pattern														
HVFD_NONE	VFD3 off														
HVFD_ENABLED	VFD3 on														
int Range	<p>Determines the length of the VFD field that will be laid into the frame.</p> <p>For VFD1 and VFD2: To specify the length in byte units, use a positive integer from 1 to 6. To specify the length in bit units, use a negative integer from -1 to -48. The minus symbol flags the library that the number represents bits instead of bytes. Since 100Mbps Ethernet cards send traffic in increments of four bits, a range that is not in multiples of four will be rounded up to the nearest nibble for these cards.</p> <p>For VFD3: The length of VFD3 is set in bytes. The byte length is from 1 to 2047.</p>														
int Offset	<p>Determines the bit number in the frame where VFD is overlaid. Measurement begins immediately after the preamble.</p> <p><i>Range:</i> 0 to 12,112.</p> <p>For a 100Mbps Ethernet SmartCard, values that are not multiples of four are rounded up to the next 4 bit (nibble) increment.</p>														

int Data	<p>Points to an array of integers that constitute the pattern for the VFD. For Visual Basic, use int*iData instead of int*Data.</p> <p>For VFD1 and VFD2 only: Elements values are entered into the array with the most significant bit first. For example:</p> <pre>iData[0] 0 iData[1] 1 iData[2] 2 iData[3] 3 iData[4] 4 iData[5] 5</pre> <p>Creates the VFD pattern: 543210[BS8]</p>
int DataCount	<p>This value has different uses for VFD1 or 2 and for VFD3.</p> <p>For VFD1 and VFD2: The DataCount is used with Configuration to limit the number of patterns generated. DataCount is the Cycle-count (number of different patterns that will be generated before being repeated).</p> <p>Example: If Configuration = HVFD_INCR And if DataCount = 6 Results in six VFD patterns. The initial pattern is used in the first frame. The next five values increment, creating a series of five new patterns. The initial pattern is then used again, and the cycle repeats itself.</p> <p>For VFD3: The buffer size of the Data array. Used in combination with the Range to determine how often a pattern is repeated. For example, if the DataCount is 24 and the Range is 6, there will be four six byte patterns before the first is repeated.</p> <p>For Gigabit Ethernet cards, the byte length is from 1 to 16384. For all other SmartCards, the byte length is from 1 to 2047[BS9].</p>

HTAlign

Description	Create alignment errors on the selected Hub/Slot/Port.
Syntax	int HTAlign(int iBits, int iHub, int iSlot, int iPort)
Parameters	<p><i>iBits</i> int Sets the number of extra alignment bits to transmit. Valid range is 0 to 7. Setting this value to 0 disables generation of packets with alignment bit errors.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in iHub) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTBurstCount

Description	Sets the number of packets to transmit in a single burst from a SmartCard.	
Syntax	int HTBurstCount(long IVal, iHub, iSlot, iPort)	
Parameters	<i>IVal</i>	long Specifies the burst count. <i>Range:</i> 1 to 16,777,215.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in Hub) to 19 (last card in Hub).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	This instruction does not cause a burst of packets to be sent. Use HGTransmitMode , or HTTransmitMode to select a burst mode, and then use HGRun , HGStart , HGStep , or HTRun to actually start the transmission of the burst.	

HTBurstGap

Description	Sets up the time gap between bursts of packets from a SmartCard.	
Syntax	int HTBurstGap(long IVal, iHub, iSlot, iPort)	
Parameters	<i>IVal</i>	long Specifies the inter-burst gap in tenths of a microsecond. <i>Range:</i> 1 to 16,777,215.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in Hub) to 19 (last card in Hub).
	<i>iPort</i>	int Identifies port on the card or module.
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	This instruction is only applied if HGTransmitMode , or HTTransmitMode has selected one of the MULTI_BURST_MODE, or CONTINUOUS_BURST mode selections. Use HGRun , HGStart , and HTRun to actually start the transmission of the bursts.	

HTBurstGapAndScale

Description	Sets up the time gap between bursts of packets, at the given scale from a SmartCard.	
Syntax	int HTBurstGapAndScale(long IVal, int iScale, iHub, iSlot, iPort)	
Parameters	<i>IVal</i>	long Specifies the inter-burst gap value. Legal values range anywhere from the lowest gap possible on the card being addressed up to a maximum of 1.6 sec.
	<i>iScale</i>	int Specifies the scale of the gap value according to following. NANO_SCALE = nanoseconds scale MICRO_SCALE = microseconds scale MILLI_SCALE = milliseconds scale Period may range from 1 to 1,600,000,000 nanoseconds for NANO_SCALE; from 1 to 1,600,000 microseconds for MICRO_SCALE; and from 1 to 1600 milliseconds for MILLI_SCALE. Values outside this range return an error code.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.	
Comments	This instruction is only applied if HGTransmitMode , or HTTransmitMode has selected one of the MULTI_BURST_MODE, or CONTINUOUS_BURST mode selections. Use HGRun , HGStart , and HTRun to actually start the transmission of the bursts.	

HTCardModel

Description	Retrieves an array of integers that corresponds to the card model written at the top of the SmartCard front panel.
Syntax	int HTCardModel(int iCardModels[MAX_HUBS][MAX_SLOTS])
Parameters	<p><i>iCardModels</i> int On return, this array will be filled with CM_ values where the hub and slot indices of the array refer to an iCardModel entry which correspond to the model of the SmartCard actually plugged into the SmartBits chassis. The returned values will be one of the following:</p> <p>CM_UNKOWN CM_NOT_PRESENT CM_SE_6205 CM_SC_6305 CM_ST_6405 CM_ST_6410 CM_SX_7205 CM_SX_7405 CM_SX_7410 CM_TR_8405 CM_VG_7605 CM_L3_6705 CM_AT_9025 CM_AT_9155 CM_AS_9155 CM_GX_1405 CM_WN_3405 CM_AT_9015 CM_AT_9020 CM_AT_9034 CM_AT_9045 CM_AT_9622 CM_L3_6710 CM_SX_7210 CM_ML_7710</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HTClearPort

Description	Clears internal counters in a SmartCard port.	
Syntax	int HTClearPort(int iHub, int iSlot, int iPort)	
Parameters	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	None	

HTCollision

Description	Determines the collision mode, and count for the 100 Mbps Fast SmartCard.	
Syntax	int HTCollision(CollisionStructure* CStruct, int iHub, int iSlot, int iPort)	
Parameters	<i>CStruct</i>	CollisionStructure* Holds information pertaining to the collision mode (off, on), and count.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See <i>Appendix B for error codes</i> .	
Comments	See the definition of CollisionStructure on page 120. The offset and length fields are not used for 100 Mbps cards.	

HTCollisionBackoffAggressiveness

Description	Determines the wait factor for backing off from multiple collisions.
Syntax	int HTCollisionBackoffAggressiveness(unsigned int uiAggressiveness, int iHub, int iSlot, int iPort)
Parameters	<p><i>uiAggressiveness</i> unsigned int Set the backoff factor. The length of time actually delayed follows as powers of two using this factor.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTCRC

Description	Create packets with CRC errors on the selected Hub/Slot/Port.
Syntax	int HTCRC(int iMode, int iHub, int iSlot, int iPort)
Parameters	<p><i>iMode</i> int Set the error facility on or off. Valid flags: ET_ON and ET_OFF</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	(Not used by the TokenRing SmartCard).

HTDataLength

Description	This command is used to specify the length of the data field in the packets being transmitted by the specified SmartBits port. A random packet size can also be selected.
Syntax	int HTDataLength(int iLength, int iHub, int iSlot, int iPort)
Parameters	<p><i>iLength</i> int Specifies the length of the packets that are to be transmitted on the addressed port. The length is specified in bytes, and it includes everything between the preamble and the CRC. The actual transmitted packet will be extended four bytes for the CRC. Length can range from 1 to 8191. A Length of 0 will cause random packet sizes to be transmitted.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTDribble

Description	Create dribble bit errors on the selected Hub/Slot/Port.
Syntax	int HTDribble(int iBits, int iHub, int iSlot, int iPort)
Parameters	<p><i>iBits</i> int Sets the number of dribble bits to transmit. Valid range is 0 to 7. Setting this value to 0 disables generation of packets with dribble bit errors. On Fast Ethernet cards and modules, this value is rounded down to 2 or 4 (nearest nibble).</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTDuplexMode

Description	Indicates whether to set full duplex or half duplex mode for the hub/slot/port indicated.	
Syntax	int HTDuplexMode(int iMode, int iHub, int iSlot, int iPort)	
Parameters	<i>iMode</i>	int Sets the Duplex mode where iMode should be one of the following: FULLDUPLEX_MODE Full duplex mode on HALFDUPLEX_MODE Half duplex mode on
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	(Not used by the TokenRing SmartCard)	

HTFillPattern

Description	Specifies the background fill pattern that is laid into the frame. This pattern is written over by other fields such as VFDs and Signature fields. If the Fill Pattern is not specified, the default is all 0s.	
Syntax	int HTFillPattern(int iSize, int* piData, int iHub, int iSlot, int iPort)	
Parameters	<i>iSize</i>	int Identifies the size, in bytes, of the fill pattern contained in the Data array. Size may range from 60 to 2044. A value of 0 (zero) will cause a random data pattern to be generated.
	<i>piData</i>	int* Points to the array which contains the data pattern to be transmitted. A value of NULL will cause a random data pattern to be generated.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	A random data pattern will be generated if either the iSize parameter is 0, or the piData array pointer parameter is NULL.	

HTFindMIIAddress

Description	This function will find the first MII PHY address that appears to have a legal device present. This command applies only to 100 Mb SmartCards.
Syntax	int HTFindMIIAddress(unsigned int* puiAddress, unsigned short* puiControl Bits, int Hub, int Slot, int Port)
Parameters	<p><i>puiAddress</i> unsigned int* Specific address found is returned here.</p> <p><i>puiControlBits</i> unsigned short* Contents of the control register are returned here.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>iHub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the SmartCard port. (On the current SmartCards, <i>Port</i> is always 0.)</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	Of the 32 possible addresses on an MII transceiver, this command will find the lowest address that returns a legal control register value.

HTFrame

Description	Puts specified frame elements into the SmartCard frame buffer. Use HTFrame in conjunction with NSCreateFrame, NSModifyFrame, and NSCreateFrameAndPayload.
Syntax	long HTFrame (long iFrameID, int iHub, int iSlot, int iPort, unsigned short uiStreamIndex)
Parameters	<p><i>iFrameID</i> long The FrameID number is unique for each frame created with NSCreateFrame. It is returned when a frame is created, and is used to identify the specified frame “blueprint”. This number does not change when NSModifyFrame is used.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p> <p><i>uiStreamIndex</i> unsigned short This uiStreamIndex has a value of 0 (not used), unless you are working with ATM. Because of the complexity of an ATM stream setup, each ATM stream must be indexed. ATM streams include traffic information as well as frame content.</p> <p>See ATM_STREAM in the Message Functions manual for more information.</p> <p>Since NSCreateFrame functions are intended for “layer 2” mode, VTEs and Signature fields are not part of these frames.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A negative value is returned if the function fails. See <i>Appendix B for error codes</i> .
Comments	A related function is NSDeleteFrame.

HTGap

Description	Specifies the inter-packet gap that is to be transmitted on the addressed port. Also allows random gaps to be transmitted.	
Syntax	int HTGap(long IPeriod, int iHub, int iSlot, int iPort)	
Parameters	<i>IPeriod</i>	long On 10Mbit cards, this value equals the number of tenths of microseconds between transmitted packets in bit time. On 100Mbit SmartCards, this value equals the number of tens of nanoseconds between transmitted packets. In either case, IPeriod may range from 10 to 1,600,000. A value of 0 (long) will cause a random gap to be generated. For example, if IPeriod = 96, for 10Mbit cards, the Gap will be $96 * 0.1\mu s = 9.6\mu s$, and for 100Mbit cards, the Gap will be $96 * 10ns = 960ns$. In both cases, the cards get the minimum legal interpacket gap.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	None	

HTGapAndScale

Description	Specifies the inter-packet gap (based on a selected time unit “scale”) to be transmitted from the specified port. Also allows random gaps to be transmitted.										
Syntax	int HTGapAndScale(long IPeriod, int iScale, int iHub, int iSlot, int iPort)										
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; vertical-align: top;"><i>IPeriod</i></td> <td>long Identifies the number of time units between transmitted packets. A value of 0 (long) will cause a random gap to be generated.</td> </tr> <tr> <td style="vertical-align: top;"><i>iScale</i></td> <td> int Determines the size of the unit (scale) for the IPeriod parameter based on the following: NANO_SCALE = nanoseconds scale MICRO_SCALE = microseconds scale MILLI_SCALE = milliseconds scale Possible values range from 1 to 1,600,000,000 nanoseconds for NANO_SCALE; from 1 to 1,600,000 microseconds for MICRO_SCALE; and from 1 to 1600 milliseconds for MILLI_SCALE. Values outside this range return an error code. </td> </tr> <tr> <td style="vertical-align: top;"><i>iHub</i></td> <td> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2. </td> </tr> <tr> <td style="vertical-align: top;"><i>iSlot</i></td> <td>int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</td> </tr> <tr> <td style="vertical-align: top;"><i>iPort</i></td> <td>int Identifies the port on the card or module.</td> </tr> </table>	<i>IPeriod</i>	long Identifies the number of time units between transmitted packets. A value of 0 (long) will cause a random gap to be generated.	<i>iScale</i>	int Determines the size of the unit (scale) for the IPeriod parameter based on the following: NANO_SCALE = nanoseconds scale MICRO_SCALE = microseconds scale MILLI_SCALE = milliseconds scale Possible values range from 1 to 1,600,000,000 nanoseconds for NANO_SCALE; from 1 to 1,600,000 microseconds for MICRO_SCALE; and from 1 to 1600 milliseconds for MILLI_SCALE. Values outside this range return an error code.	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).	<i>iPort</i>	int Identifies the port on the card or module.
<i>IPeriod</i>	long Identifies the number of time units between transmitted packets. A value of 0 (long) will cause a random gap to be generated.										
<i>iScale</i>	int Determines the size of the unit (scale) for the IPeriod parameter based on the following: NANO_SCALE = nanoseconds scale MICRO_SCALE = microseconds scale MILLI_SCALE = milliseconds scale Possible values range from 1 to 1,600,000,000 nanoseconds for NANO_SCALE; from 1 to 1,600,000 microseconds for MICRO_SCALE; and from 1 to 1600 milliseconds for MILLI_SCALE. Values outside this range return an error code.										
<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.										
<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).										
<i>iPort</i>	int Identifies the port on the card or module.										
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .										
Comments	Gap is set according to the valid increments of the network topography. For example, if a 100 Mbps Ethernet network is being tested, the gap is set in increments of 40 nanoseconds. Whether nanoseconds, microseconds, or milliseconds is selected, SmartLib divides the increment (in this case, 40 ns) into the desired gap setting and drops the remainder.										

HTGetCardModel

Description	Retrieves a character string which matches the card model written at the top of the SmartCard front panel.																																																																
Syntax	int HTGetCardModel(char* pszCardModel, int iHub, int iSlot, int iPort)																																																																
Parameters	<p><i>pszCardModel</i> char* A pointer to a character array into which the card model identifier will be written. The card model identifier is the front panel label on the SmartCard (e.g. L3-6710, ML-7710, AT-9622, etc).</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>																																																																
Return Value	<p>Upon success, the return value is the correct CM_ integer value for the SmartCard addressed. Possible values are:</p> <table border="0"> <tr><td>CM_UNKNOWN</td><td>-1</td></tr> <tr><td>CM_NOT_PRESENT</td><td>0</td></tr> <tr><td>CM_SE_6205</td><td>1</td></tr> <tr><td>CM_SC_6305</td><td>2</td></tr> <tr><td>CM_ST_6405</td><td>3</td></tr> <tr><td>CM_ST_6410</td><td>4</td></tr> <tr><td>CM_SX_7205</td><td>5</td></tr> <tr><td>CM_SX_7405</td><td>6</td></tr> <tr><td>CM_SX_7410</td><td>7</td></tr> <tr><td>CM_TR_8405</td><td>8</td></tr> <tr><td>CM_VG_7605</td><td>9</td></tr> <tr><td>CM_L3_6705</td><td>10</td></tr> <tr><td>CM_AT_9025</td><td>11</td></tr> <tr><td>CM_AT_9155</td><td>12</td></tr> <tr><td>CM_AS_9155</td><td>13</td></tr> <tr><td>CM_GX_1405</td><td>14</td></tr> <tr><td>CM_WN_3405</td><td>15</td></tr> <tr><td>CM_AT_9015</td><td>16</td></tr> <tr><td>CM_AT_9020</td><td>17</td></tr> <tr><td>CM_AT_9034</td><td>18</td></tr> <tr><td>CM_AT_9045</td><td>19</td></tr> <tr><td>CM_AT_9622</td><td>20</td></tr> <tr><td>CM_L3_6710</td><td>21</td></tr> <tr><td>CM_SX_7210</td><td>22</td></tr> <tr><td>CM_ML_7710</td><td>23</td></tr> <tr><td>CM_ML_5710A</td><td>24</td></tr> <tr><td>CM_WN_3415</td><td>25</td></tr> <tr><td>CM_WN_3420</td><td>26</td></tr> <tr><td>CM_LAN-6200</td><td>27</td></tr> <tr><td>CM_LAN-6100</td><td>28</td></tr> <tr><td>CM_LAN_3200</td><td>29</td></tr> <tr><td>CM_POS_6500</td><td>30</td></tr> </table>	CM_UNKNOWN	-1	CM_NOT_PRESENT	0	CM_SE_6205	1	CM_SC_6305	2	CM_ST_6405	3	CM_ST_6410	4	CM_SX_7205	5	CM_SX_7405	6	CM_SX_7410	7	CM_TR_8405	8	CM_VG_7605	9	CM_L3_6705	10	CM_AT_9025	11	CM_AT_9155	12	CM_AS_9155	13	CM_GX_1405	14	CM_WN_3405	15	CM_AT_9015	16	CM_AT_9020	17	CM_AT_9034	18	CM_AT_9045	19	CM_AT_9622	20	CM_L3_6710	21	CM_SX_7210	22	CM_ML_7710	23	CM_ML_5710A	24	CM_WN_3415	25	CM_WN_3420	26	CM_LAN-6200	27	CM_LAN-6100	28	CM_LAN_3200	29	CM_POS_6500	30
CM_UNKNOWN	-1																																																																
CM_NOT_PRESENT	0																																																																
CM_SE_6205	1																																																																
CM_SC_6305	2																																																																
CM_ST_6405	3																																																																
CM_ST_6410	4																																																																
CM_SX_7205	5																																																																
CM_SX_7405	6																																																																
CM_SX_7410	7																																																																
CM_TR_8405	8																																																																
CM_VG_7605	9																																																																
CM_L3_6705	10																																																																
CM_AT_9025	11																																																																
CM_AT_9155	12																																																																
CM_AS_9155	13																																																																
CM_GX_1405	14																																																																
CM_WN_3405	15																																																																
CM_AT_9015	16																																																																
CM_AT_9020	17																																																																
CM_AT_9034	18																																																																
CM_AT_9045	19																																																																
CM_AT_9622	20																																																																
CM_L3_6710	21																																																																
CM_SX_7210	22																																																																
CM_ML_7710	23																																																																
CM_ML_5710A	24																																																																
CM_WN_3415	25																																																																
CM_WN_3420	26																																																																
CM_LAN-6200	27																																																																
CM_LAN-6100	28																																																																
CM_LAN_3200	29																																																																
CM_POS_6500	30																																																																
Return Value	<table border="0"> <tr><td>CM_GX_1420</td><td>31</td></tr> <tr><td>CM_LAN-6201</td><td>32</td></tr> <tr><td>CM_AT_9155C</td><td>33</td></tr> </table> <p>A failure code (less than zero) is returned if the function failed. See <i>Appendix B</i> for error codes.</p>	CM_GX_1420	31	CM_LAN-6201	32	CM_AT_9155C	33																																																										
CM_GX_1420	31																																																																
CM_LAN-6201	32																																																																
CM_AT_9155C	33																																																																
Comments	None																																																																

HTGetCounters

Description	Retrieves information from all the counters within the addressed SmartBits port. This information is placed into the HTCountStructure pointed to in the input argument.
Syntax	int HTGetCounters (HTCountStructure* pHTHStruct, int iHub, int iSlot, int iPort)
Parameters	<p><i>pHTHStruct</i> HTCountStructure* A pointer to the structure in which count information is to be placed. See the definition of HTCountStructure below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	It is assumed that the calling function has declared a HTCountStructure and reserved memory for it.

HTCountStructure

Parameter	Description
unsigned long RcvPkt	Current number of packets received
unsigned long TmtPkt	Current number of packets transmitted
unsigned long Collision	Current number of collisions
unsigned long RcvTrig	Current number of Trigger received
unsigned long RcvByte	Current number of Bytes received
unsigned long CRC	Current number of CRC errors received
unsigned long Align	Current number of Alignment errors detected
unsigned long Oversize	Current number of Oversize errors detected
unsigned long Undersize	Current number of Undersize errors detected
unsigned long RcvPktRate	Number of received packets per second
unsigned long TmtPktRate	Number of transmitted packets per second
unsigned long CRCRate	Number of CRC errors received per second
unsigned long OversizeRate	Number of Oversize errors received per second
unsigned long UndersizeRate	Number of Undersize errors received per second
unsigned long CollisionRate	Number of Collisions detected per second
unsigned long AlignRate	Number of Alignment errors received per second
unsigned long RcvTrigRate	Number of triggers received per second
unsigned long RcvByteRate	Number of bytes received per second

HTGetEnhancedCounters

Description	Retrieves standard counters and card related counters from the port. This information is placed into the <code>EnhancedCountersStructure</code> pointed to in the input argument.	
Syntax	<code>int HTGetEnhancedCounters(EnhancedCountersStructure* pEnCounter, int iHub, int iSlot, int iPort)</code>	
Parameters	<i>pEnCounter</i>	EnhancedCountersStructure* A pointer to the first element of an array of counter structures in which count information is to be placed. See the definition of EnhancedCountersStructure on page 125.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	None	

HTGetEnhancedStatus

Description	Retrieves card related status information from the port. This information is placed into the long pointed to in the input argument. This command applies to SmartCards and TokenRing SmartCards.	
Syntax	<code>int HTGetEnhancedStatus(long* piData, int iHub, int iSlot, int iPort)</code>	
Parameters	<i>piData</i>	long* A pointer to an long in which status information is to be placed.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> . If the return is successful, then the following is true: <i>piData</i> A bitmap of card status information is returned in the high three bytes: If bit set on a Token Ring SmartCard: <pre> TR_STATUS_ACCESSED Card received stream download TR_STATUS_BADSTREAM Not used TR_STATUS_BURST_MODE Card is in burst mode TR_STATUS_BEACONING Card received MAC beacon frame TR_STATUS_DEVICE If set in half duplex, station If off in half duplex, MAU If set in full duplex, adapter If off in full duplex, concentrator </pre>	

	<pre> TR_STATUS_EARLY_TOKEN_RELEASE Early token release enabled TR_STATUS_FULL_DUPLEX Full duplex TR_STATUS_16MB 16 Mbps mode TR_STATUS_RING_ALIVE Ready for TX TR_STATUS_LATENCY_STABLE Latency value stable for readout. TR_STATUS_TRANSMITTING Transmitting If bit set on a Gigabit Ethernet SmartCard: GIG_STATUS_LINK Link established GIG_STATUS_TX_PAUSE Pause holdoff in process GIG_STATUS_CAPTURED_FRAMES Frames captured GIG_STATUS_CAPTURE_STOPPED Capture stopped If bit set on an SX-7410 SmartCard: FAST7410_STATUS_LINK Link established FAST7410_STATUS_TX_PAUSE Pause holdoff in process If bit set on a L3-6705 or L3-6710 SmartCard: L3_STATUS_6710 If set, L3-6710, If off, L3-6705 If bit set on a VG-xxxx SmartCard: VG_STATUS_MODE If set, Ethernet, If off, TokenRing If bit set on a Frame Relay SmartCard: FR_STATUS_LINK_OK link established FR_STATUS_GROUP_MEMBER card is "grouped" FR_STATUS_UNI_UP UNI is up FR_STATUS_EIA_DSR DSR line is high FR_STATUS_EIA_CTS CTS line is high FR_STATUS_EIA_DCD DCD line is high FR_STATUS_EIA_TM TM line is high FR_STATUS_EIA_DTR DTR line is high FR_STATUS_EIA_RTS RTS line is high FR_STATUS_EIA_RDL RDL line is high FR_STATUS_EIA_LLB LLB line is high </pre>
Comments	The low byte contains card LED information. Refer to the appendix on LED values for more information.

HTGetHubLEDs

Description	Determine the state of the LEDs on a SmartBits hub.
Syntax	int HTGetHubLEDs(int iHub, int* piLEDs)
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>piLEDs</i> int* a pointer to an integer array of MAX_SLOTS size that receives the LED states of all SmartCards in hub iHub.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	Behavior of this function is undefined if the hub contains passive cards.

HTGetLEDs

Description	Determine the state of the LEDs on an SmartCard type at the specified hub/slot/port.												
Syntax	int HTGetLEDs(int iHub, int iSlot, int iPort)												
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs) – 1. Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>												
Return Value	<p>The return value is the current state of the LEDs. This return value can be ANDed against the following to determine if the LED is on.</p> <table border="0"> <tr> <td>HTLED_TXRED</td> <td>Unconfigured card</td> </tr> <tr> <td>HTLED_TXGREEN</td> <td>Transmitting</td> </tr> <tr> <td>HTLED_COLLRED</td> <td>Collision detected</td> </tr> <tr> <td>HTLED_COLLGREEN</td> <td>Trigger detected</td> </tr> <tr> <td>HTLED_RXRED</td> <td>Receive with errors</td> </tr> <tr> <td>HTLED_RXGREEN</td> <td>Receive</td> </tr> </table> <p>A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i>.</p>	HTLED_TXRED	Unconfigured card	HTLED_TXGREEN	Transmitting	HTLED_COLLRED	Collision detected	HTLED_COLLGREEN	Trigger detected	HTLED_RXRED	Receive with errors	HTLED_RXGREEN	Receive
HTLED_TXRED	Unconfigured card												
HTLED_TXGREEN	Transmitting												
HTLED_COLLRED	Collision detected												
HTLED_COLLGREEN	Trigger detected												
HTLED_RXRED	Receive with errors												
HTLED_RXGREEN	Receive												
Comments	This function is available only for SmartCards. LED return states are not a hardware function, but are derived from the states of the counters. If both HTLED_COLLRED and HTLED_COLLGREEN are set, then the LED is yellow. No other LED can be yellow.												

HTGetHWVersion

Description	Retrieves version information of the specified SmartCard. Information is retrieved into <i>pulData</i> .								
Syntax	int HTGetHWVersion(unsigned long* pulData, int iHub, int iSlot, int iPort)								
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"><i>pulData</i></td> <td>unsigned long* A pointer to an unsigned long array in which version information is to be placed. The size of the array depends on specific card inquired. An array size of 32 is recommended. [See comments below.]</td> </tr> <tr> <td><i>iHub</i></td> <td>int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</td> </tr> <tr> <td><i>iSlot</i></td> <td>int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</td> </tr> <tr> <td><i>iPort</i></td> <td>int Identifies the port on the card or module.</td> </tr> </table>	<i>pulData</i>	unsigned long* A pointer to an unsigned long array in which version information is to be placed. The size of the array depends on specific card inquired. An array size of 32 is recommended. [See comments below.]	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).	<i>iPort</i>	int Identifies the port on the card or module.
<i>pulData</i>	unsigned long* A pointer to an unsigned long array in which version information is to be placed. The size of the array depends on specific card inquired. An array size of 32 is recommended. [See comments below.]								
<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.								
<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).								
<i>iPort</i>	int Identifies the port on the card or module.								
Return Value	The return value is ≥ 0 if the function executed successfully and will indicate the number of items in the <i>pulData</i> array which have been loaded with version information related to this SmartCard. For example, a TokenRing Card will return 3. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .								
Comments	Each SmartCard will fill the <i>pulData</i> array with only that number of items that is given as the return value. No other items in the <i>pulData</i> will be changed. A TokenRing Card will return Firmware, Transmit, and Receive information in the unsigned long array pointed at by <i>pulData</i> . It is recommended to zero the <i>pulData</i> array items prior to this call.								

HTGetStructure

Description	Sends a command to a SmartCard which accepts HTGetStructure() actions. The commands, defines, and structure definitions for this function can be found in the <i>Message Functions</i> manual. See the separate sections for Ethernet, Fast Ethernet, Gigabit, Multi-Layer, ATM, and Frame Relay SmartCards. These SmartCards allow control using HTSetCommand(), HTSetStructure(), and HTGetStructure(). The correct combination of iType parameter values and the structure parameter cause the SmartCards to be set up in an elegant and intricate manner.
Syntax	int HTGetStructure(int iType1,int iType2,int iType3,int iType4,void* pData,int iSize,int iHub, int iSlot, int iPort);
Parameters	<p><i>iType1</i> int defines the command action. The value (and action) depends on the SmartCard being addressed.</p> <p><i>iType2</i> int value depends on SmartCard</p> <p><i>iType3</i> int value depends on SmartCard</p> <p><i>iType4</i> int value depends on SmartCard</p> <p><i>pData</i> void* Pointer to a structure or an array in which returned data will be placed.</p> <p><i>iSize</i> int indicates the maximum size of the pData pointer which should be utilized. While in most cases this will be the size of the structure, in some cases it is the size of an array of structures or bytes. See the <i>Message functions</i> manual for clarification.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the SmartCard is located. <i>Range:</i> 0 (first slot in Hub) to 19 (last card in Hub).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is >= 0 if the function executed successfully. The exact value will vary according to what iType parameters have been used The return value is < 0 if the function failed. See <i>Appendix B for error codes</i> .
Comments	See the <i>Message functions</i> manual for appropriate values for the iType and structure parameters for HTSetCommand(), HTSetStructure(), and HTGetStructure().

HTHubId

Description	Fill an array with the currently connected port types.																
Syntax	int HTHubId(char PortTypes[MAX_HUBS][MAX_SLOTS][MAX_PORTS])																
Parameters	<p><i>PortTypes</i> char An array of character that will be filled with one of the available card types. The card types are:</p> <table style="margin-left: 40px;"> <tr><td>A</td><td>10Mb Ethernet</td></tr> <tr><td>F</td><td>10/100Mb Fast Ethernet</td></tr> <tr><td>T</td><td>4/16Mb TokenRing</td></tr> <tr><td>V</td><td>VG/AnyLan</td></tr> <tr><td>3</td><td>Layer 3 10Mb Ethernet</td></tr> <tr><td>G</td><td>Gigabit Ethernet</td></tr> <tr><td>S</td><td>ATM Signaling</td></tr> <tr><td>N</td><td>Not present</td></tr> </table>	A	10Mb Ethernet	F	10/100Mb Fast Ethernet	T	4/16Mb TokenRing	V	VG/AnyLan	3	Layer 3 10Mb Ethernet	G	Gigabit Ethernet	S	ATM Signaling	N	Not present
A	10Mb Ethernet																
F	10/100Mb Fast Ethernet																
T	4/16Mb TokenRing																
V	VG/AnyLan																
3	Layer 3 10Mb Ethernet																
G	Gigabit Ethernet																
S	ATM Signaling																
N	Not present																
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>																
Comments	None																

HTHubSlotPorts

Description	Fill an array with the currently connected port types.																
Syntax	int HTHubSlotPorts(int iPortTypes[MAX_HUBS][MAX_SLOTS][MAX_PORTS])																
Parameters	<p><i>iPortTypes</i> int An array of integers that will be filled with one of the available card types. The card types are:</p> <table style="margin-left: 40px;"> <tr><td>CT_ACTIVE</td><td>10Mb Ethernet</td></tr> <tr><td>CT_FASTX</td><td>10/100Mb Ethernet</td></tr> <tr><td>CT_TOKENRING</td><td>4/16Mb TokenRing</td></tr> <tr><td>CT_VG</td><td>VG/AnyLan</td></tr> <tr><td>CT_L3</td><td>Layer 3 10Mb Ethernet</td></tr> <tr><td>CT_GIGABIT</td><td>Gigabit Ethernet</td></tr> <tr><td>CT_ATM_SIGNALING</td><td>ATM Signaling</td></tr> <tr><td>CT_NOT_PRESENT</td><td>Not present</td></tr> </table>	CT_ACTIVE	10Mb Ethernet	CT_FASTX	10/100Mb Ethernet	CT_TOKENRING	4/16Mb TokenRing	CT_VG	VG/AnyLan	CT_L3	Layer 3 10Mb Ethernet	CT_GIGABIT	Gigabit Ethernet	CT_ATM_SIGNALING	ATM Signaling	CT_NOT_PRESENT	Not present
CT_ACTIVE	10Mb Ethernet																
CT_FASTX	10/100Mb Ethernet																
CT_TOKENRING	4/16Mb TokenRing																
CT_VG	VG/AnyLan																
CT_L3	Layer 3 10Mb Ethernet																
CT_GIGABIT	Gigabit Ethernet																
CT_ATM_SIGNALING	ATM Signaling																
CT_NOT_PRESENT	Not present																
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>																
Comments	For Tcl: Use the utility function ETMake3DArray in order to create 3D arrays in Tcl.																

HTLatency

Description	Tests latency using the SmartBits.
Syntax	int HTLatency(int iMode, HTLatencyStructure* pHTLat, int iHub, int iSlot, int iPort)
Parameters	<p><i>iMode</i> int Set one of four specific modes of operation:</p> <p> HT_LATENCY_OFF removes the SmartCard from participating in any latency measurements.</p> <p> HT_LATENCY_RX Sets the SmartCard as a latency receiver. Only ports set as receivers can use the latency report function.</p> <p> HT_LATENCY_RXTX Set as latency receiver, and also as latency transmitter. The receive setting enables the latency report function on this card</p> <p> HT_LATENCY_TX Set as latency transmitter. (can not use the latency report function)</p> <p> HT_LATENCY_REPORT Enables latency counter value to be returned in the ulReport member of the HTLatencyStructure provided in pHTLat below. (The Latency Counter value is in units of 100 nanoseconds.) Only ports set as receivers will obtain valid results when using this mode. The latency counters start running when a group transmit function starts, and stops when a packet matching the contents and at the position of data set in pHTLat.</p> <p><i>pHTLat</i> HTLatencyStructure* This structure sets the position, size and contents of packet data that will stop latency counters when a complete match occurs, and holds the ulReport value when retrieving the latency measurements on each port. See the definition of HTLatencyStructure below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one hub IDs start at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>Note - When using this command, VFD 3 of the transmitting port, and the Triggers of any receiving ports are utilized. [Also, on 10MB cards, the ByteCount counter function is disabled.]</p> <p>The latency counter is a special counter in a SmartCard. It is enabled when a card is set in latency mode, and starts counting when a group transmit command [e.g. HGGroupStart()] is issued. It stops when a packet is received which matches the characteristics specified in the HTLatencyStructure when HT_LATENCY_RX or HT_LATENCY_RXTX was issued.</p> <p>The actual latency measurement is determined by subtracting the HTLatencyStructure.ulReport values of the transmitting SmartCard from the receiver SmartCard. This difference is the bit to bit latency measurement. Your program will need to make any adjustments for cut-through vs. store and forward operations of the device(s) attached to each port.</p> <p>On 10MB cards, the ByteCount counter function is superseded with the Latency counter function. When getting the counters from a 10MB card that is included in a Latency measurement, the ByteCount value will reflect the raw Latency measurement.</p>

HTLatencyStructure

Parameter	Description
int Range	Size of the iData array to use, in bytes.
int Offset	Offset in bits for the first bit of the iData trigger from the first bit of the transmitted packet.
int iData[12]	Actual data that will stop the latency counter.
unsigned long uLatency	Receives the latency value when using HT_LATENCY_REPORT. See the HTLatency function for more details.

HTLayer3SetAddress

Description	Configures the card to send/receive background traffic such as PING, SNMP, etc. This command is not used to set up regular L3 test streams.
Syntax	int HTLayer3SetAddress (Layer3Address* pLayer3Address, int iHub, int iSlot, int iPort)
Parameters	<p><i>pLayer3Address</i> Layer3Address A pointer to the structure containing Layer 3 information such as IP address. See the definition of Layer3Address below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. <i>Range</i>: 0 (first hub) through N (number of hubs – 1). Remember to subtract one; hub IDs starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the destination SmartCard.</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	<p>Use HTLayer3SetAddress if you want to send additional frames during your test process such as PING, SNMP, RIP, and Card ARP response.</p> <p>This command is not necessary for defining test traffic. To set up test traffic (traditional mode) see the NSCreateFrame series. To set up test traffic in the more powerful SmartMetrics mode, see the Message Functions manual under your specific SmartCard type.</p> <p>For using this command with multiple SmartCards in Tcl see also: ETMake3DArray.</p>

Layer3Address

Use this structure with the HTLayer3SetAddress function to set background traffic in addition to the defined test streams. (Refer to the *Message Functions* manual for the steps to create Layer 3 streams.)

Parameter	Description
unsigned char szMACAddress[6]	Sets MAC address of the card.
unsigned char IP[4]	Sets IP address of the card.
unsigned char Netmask[4]	Sets Netmask for the card.
unsigned char Gateway[4]	Sets Gateway address for the card.
unsigned char TargetAddress[4]	Address to which PING and SNMP frames are sent.

Parameter	Description
int iControl	<p>L3_CTRL_ARP_RESPONSES Enables responses to all received ARP requests by Tx of ARP response frames. <i>Use with caution; not recommended for online testing.</i></p> <p>L3_CTRL_PING_RESPONSES Enables Tx of PING frames.</p> <p>L3_CTRL_SNMP_OR_RIP_RESPONSES Enables Tx of SNMP/RIP frames.</p> <p>The parameters below set the intervals at which frames are sent.</p>
int iPingTime	How often (in seconds) a PING frame is transmitted. 0 = no PING frames.
int iSNMPTime	How often (in seconds) an SNMP frame is transmitted. 0 = no SNMP frames.
int iRIPTime	How often (in seconds) a RIP frame is transmitted. 0 = no RIP frames.
int iGeneralARPResponse	<i>Obsolete.</i>

HTMultiBurstCount

Description	Sets up the number of bursts for transmitting out a SmartCard while in MULTI_BURST_MODE.
Syntax	int HTMultiBurstCount(long IVal, iHub, iSlot, iPort)
Parameters	<p><i>IVal</i> long Specifies the burst count. <i>Range:</i> 1 to 16,777,215.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	This instruction is only applied if HGTransmitMode , or HTTransmitMode has selected MULTI_BURST_MODE. Use HGRun , HGStart , and HTRun to start the transmission of the bursts.

HTPPortProperty

Description	Determine the card type at the specified hub/slot/port.																																																										
Syntax	int HTPortProperty(unsigned long *pulProperties, int iHub, int iSlot, int iPort)																																																										
Parameters	<p><i>pulProperties</i> unsigned long * The contents of this address gets filled with the value of the properties for the specified port. The value is filled with the logical OR values below. This value can be ANDed against the following to determine if the Port Property is present:</p> <table border="0"> <tr><td>CA_SIGNALRATE_10MB</td><td>10MB capable</td></tr> <tr><td>CA_SIGNALRATE_100MB</td><td>100MB capable</td></tr> <tr><td>CA_DUPLEX_FULL</td><td>Full Duplex capable</td></tr> <tr><td>CA_DUPLEX_HALF</td><td>Half Duplex capable</td></tr> <tr><td>CA_CONNECT_MII</td><td>MMI connector</td></tr> <tr><td>CA_CONNECT_TP</td><td>Twisted Pair connector</td></tr> <tr><td>CA_CONNECT_BNC</td><td>BNC connector</td></tr> <tr><td>CA_CONNECT_AUI</td><td>AUI connector</td></tr> <tr><td>CA_CAN_ROUTE</td><td>Routing capable</td></tr> <tr><td>CA_VFDRESETCOUNT</td><td>Resets VFD1 &2 counter</td></tr> <tr><td>CA_SIGNALRATE_4MB</td><td>4MB capable</td></tr> <tr><td>CA_SIGNALRATE_16MB</td><td>16MB capable</td></tr> <tr><td>CA_CAN_COLLIDE</td><td>Generates collisions</td></tr> <tr><td>CA_SIGNALRATE_25MB</td><td>25MB capable</td></tr> <tr><td>CA_SIGNALRATE_155MB</td><td>155MB capable</td></tr> <tr><td>CA_BUILT_IN_ADDRESS</td><td>Has a built in address</td></tr> <tr><td>CA_HAS_DEBUG_MONITOR</td><td>Allows Debug monitoring</td></tr> <tr><td>CA_SIGNALRATE_1000MB</td><td>1 GB capable</td></tr> <tr><td>CA_CONNECT_FIBER</td><td>Fiber optic connector</td></tr> <tr><td>CA_CAN_CAPTURE</td><td>Has capture capability</td></tr> <tr><td>CA_ATM_SIGNALING</td><td>Performs ATM Signaling</td></tr> <tr><td>CA_CONNECT_V35</td><td></td></tr> <tr><td>CA_SIGNALRATE_8MB</td><td></td></tr> <tr><td>CA_SIGNALRATE_622MB</td><td></td></tr> <tr><td>CA_SIGNALRATE_45MB</td><td></td></tr> <tr><td>CA_SIGNALRATE_34MB</td><td></td></tr> <tr><td>CA_SIGNALRATE_1_544MB</td><td></td></tr> <tr><td>CA_SIGNALRATE_2_048MB</td><td></td></tr> <tr><td>CA_HASVFDREPEATCOUNT</td><td></td></tr> </table> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>	CA_SIGNALRATE_10MB	10MB capable	CA_SIGNALRATE_100MB	100MB capable	CA_DUPLEX_FULL	Full Duplex capable	CA_DUPLEX_HALF	Half Duplex capable	CA_CONNECT_MII	MMI connector	CA_CONNECT_TP	Twisted Pair connector	CA_CONNECT_BNC	BNC connector	CA_CONNECT_AUI	AUI connector	CA_CAN_ROUTE	Routing capable	CA_VFDRESETCOUNT	Resets VFD1 &2 counter	CA_SIGNALRATE_4MB	4MB capable	CA_SIGNALRATE_16MB	16MB capable	CA_CAN_COLLIDE	Generates collisions	CA_SIGNALRATE_25MB	25MB capable	CA_SIGNALRATE_155MB	155MB capable	CA_BUILT_IN_ADDRESS	Has a built in address	CA_HAS_DEBUG_MONITOR	Allows Debug monitoring	CA_SIGNALRATE_1000MB	1 GB capable	CA_CONNECT_FIBER	Fiber optic connector	CA_CAN_CAPTURE	Has capture capability	CA_ATM_SIGNALING	Performs ATM Signaling	CA_CONNECT_V35		CA_SIGNALRATE_8MB		CA_SIGNALRATE_622MB		CA_SIGNALRATE_45MB		CA_SIGNALRATE_34MB		CA_SIGNALRATE_1_544MB		CA_SIGNALRATE_2_048MB		CA_HASVFDREPEATCOUNT	
CA_SIGNALRATE_10MB	10MB capable																																																										
CA_SIGNALRATE_100MB	100MB capable																																																										
CA_DUPLEX_FULL	Full Duplex capable																																																										
CA_DUPLEX_HALF	Half Duplex capable																																																										
CA_CONNECT_MII	MMI connector																																																										
CA_CONNECT_TP	Twisted Pair connector																																																										
CA_CONNECT_BNC	BNC connector																																																										
CA_CONNECT_AUI	AUI connector																																																										
CA_CAN_ROUTE	Routing capable																																																										
CA_VFDRESETCOUNT	Resets VFD1 &2 counter																																																										
CA_SIGNALRATE_4MB	4MB capable																																																										
CA_SIGNALRATE_16MB	16MB capable																																																										
CA_CAN_COLLIDE	Generates collisions																																																										
CA_SIGNALRATE_25MB	25MB capable																																																										
CA_SIGNALRATE_155MB	155MB capable																																																										
CA_BUILT_IN_ADDRESS	Has a built in address																																																										
CA_HAS_DEBUG_MONITOR	Allows Debug monitoring																																																										
CA_SIGNALRATE_1000MB	1 GB capable																																																										
CA_CONNECT_FIBER	Fiber optic connector																																																										
CA_CAN_CAPTURE	Has capture capability																																																										
CA_ATM_SIGNALING	Performs ATM Signaling																																																										
CA_CONNECT_V35																																																											
CA_SIGNALRATE_8MB																																																											
CA_SIGNALRATE_622MB																																																											
CA_SIGNALRATE_45MB																																																											
CA_SIGNALRATE_34MB																																																											
CA_SIGNALRATE_1_544MB																																																											
CA_SIGNALRATE_2_048MB																																																											
CA_HASVFDREPEATCOUNT																																																											

Return Value	<p>The return value is one of the following if the function executed successfully:</p> <table border="0"> <tr><td>CT_NOT_PRESENT</td><td>0 (Card not present)</td></tr> <tr><td>CT_ACTIVE</td><td>1</td></tr> <tr><td>CT_FASTX</td><td>3</td></tr> <tr><td>CT_TOKENRING</td><td>4</td></tr> <tr><td>CT_VG</td><td>5</td></tr> <tr><td>CT_GIGABIT</td><td>8</td></tr> <tr><td>CT_ATM_SIGNALING</td><td>9</td></tr> <tr><td>CT_WAN_FRAME_RELAY</td><td>10</td></tr> <tr><td>CT_MAX_CARD_TYPE</td><td>CT_WAN_FRAME_RELAY</td></tr> </table> <p>A failure code of less than zero is returned if the function failed. See Appendix B for error codes.</p>	CT_NOT_PRESENT	0 (Card not present)	CT_ACTIVE	1	CT_FASTX	3	CT_TOKENRING	4	CT_VG	5	CT_GIGABIT	8	CT_ATM_SIGNALING	9	CT_WAN_FRAME_RELAY	10	CT_MAX_CARD_TYPE	CT_WAN_FRAME_RELAY
CT_NOT_PRESENT	0 (Card not present)																		
CT_ACTIVE	1																		
CT_FASTX	3																		
CT_TOKENRING	4																		
CT_VG	5																		
CT_GIGABIT	8																		
CT_ATM_SIGNALING	9																		
CT_WAN_FRAME_RELAY	10																		
CT_MAX_CARD_TYPE	CT_WAN_FRAME_RELAY																		
Comments	For more detail about CA_VFDRESETCOUNT, see HT_VFD_Structure.																		

HTPortType

Description	Determine the card type at the specified hub/slot/port.																		
Syntax	int HTPortType(int iHub, int iSlot, int iPort)																		
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>																		
Return Value	<p>The return value is one of the following if the function executed successfully:</p> <table border="0"> <tr><td>CT_NOT_PRESENT</td><td>0 (Card not present)</td></tr> <tr><td>CT_ACTIVE</td><td>1</td></tr> <tr><td>CT_FASTX</td><td>3</td></tr> <tr><td>CT_TOKENRING</td><td>4</td></tr> <tr><td>CT_VG</td><td>5</td></tr> <tr><td>CT_GIGABIT</td><td>8</td></tr> <tr><td>CT_ATM_SIGNALING</td><td>9</td></tr> <tr><td>CT_WAN_FRAME_RELAY</td><td>10</td></tr> <tr><td>CT_MAX_CARD_TYPE</td><td>CT_WAN_FRAME_RELAY</td></tr> </table> <p>A failure code of less than zero is returned if the function failed. See Appendix B for error codes.</p>	CT_NOT_PRESENT	0 (Card not present)	CT_ACTIVE	1	CT_FASTX	3	CT_TOKENRING	4	CT_VG	5	CT_GIGABIT	8	CT_ATM_SIGNALING	9	CT_WAN_FRAME_RELAY	10	CT_MAX_CARD_TYPE	CT_WAN_FRAME_RELAY
CT_NOT_PRESENT	0 (Card not present)																		
CT_ACTIVE	1																		
CT_FASTX	3																		
CT_TOKENRING	4																		
CT_VG	5																		
CT_GIGABIT	8																		
CT_ATM_SIGNALING	9																		
CT_WAN_FRAME_RELAY	10																		
CT_MAX_CARD_TYPE	CT_WAN_FRAME_RELAY																		
Comments	None																		

HTReadMII

Description	Reads a specific MII Address/Register. This command applies only to 100Mbps cards.
Syntax	int HTReadMII(unsigned int uiAddress, unsigned int uiRegister, unsigned short* puiBits, int iHub, int iSlot, int iPort)
Parameters	<p><i>uiAddress</i> unsigned int Specific address. Must be from 0 to 31</p> <p><i>uiRegister</i> unsigned int Specific register. Must be from 0 to 31</p> <p><i>puiBits</i> unsigned short* Bits read are returned here</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTResetPort

Description	Resets the addressed Card to a default condition with all errors off.
Syntax	int HTResetPort(int iResetType, int iHub, int iSlot, int iPort)
Parameters	<p><i>iResetType</i> int Identifies the run mode of the board. Legal modes can be conveyed using the following constants:</p> <p> RESET_FULL Reset all card parameters including hardware interface parameters (e.g. Token Ring Speed)</p> <p> RESET_PARTIAL Reset all card parameters except hardware interface parameters. This option can be used for Token Ring cards, to keep the card in the ring.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	This command is not implemented on the ATM and WAN (FR) card families at this time.

HTRun

Description	Sets up the run state of a card. A card can be set up to transmit a series of packets (“RUN” state), transmit a single packet (“STEP” state) or stop transmission altogether (“STOP” state). If the Burst mode has been set up to transmit a burst of packets (using the HTTransmitMode command), then transitioning from “STOP” to “RUN” will cause the specified number of packets to be transmitted.
Syntax	int HTRun(int Mode, int iHub, int iSlot, int iPort)
Parameters	<p><i>Mode</i> int Identifies the run mode of the board. Legal modes can be conveyed using the following constants:</p> <p>HTRUN **For Visual Basic use HTRUN_VALUE. **Transmit continuously or send a burst of packets.</p> <p>HTSTEP Transmit a single packet.</p> <p>HTSTOP Halt transmission of packets.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	<p>Because Visual Basic does not distinguish by case, these values have been put in the ETSMBAPI.TXT file to be used for the Mode parameter:</p> <pre>HTRUN_VALUE Transmit continuously or send a burst of packets.</pre> <p>Note: Select a desired mode using HTTransmitMode before using the HTRUN function. Otherwise the transmit mode will be the one used previously.</p>

HTSelectReceive

Description	Selects a port on a SmartBits that is to be used for receive data. The receive data from this port is routed directly back to the SmartBits’s Port B for detailed analysis.
Syntax	int HTSelectReceive(int iHub, int iSlot, int iPort)
Parameters	<p><i>iHub</i> int Identifies the destination hub where the card is located. Can range anywhere from 0 (first hub) to 3 (fourth hub).</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port of the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	<p>If any of iHub, iSlot, iPort are equal to –1, the last selected port will be disabled.</p> <p>If disabling HTSelectReceive and the last selected port is unknown, then the first available active port will be selected, then deselected. No check is made as to whether this card is currently transmitting. This function assumes that at least one other SmartBits is attached to the SmartBits. It will be ignored by the SmartBits if there is not another SmartBits present.</p>

HTSelectTransmit

Description	Enables the Port B transmission of the SmartBits to be transmitted to the port specified. Transmission mode is determined by <i>iMode</i> .
Syntax	int HTSelectTransmit(int iMode, int iHub, int iSlot, int iPort)
Parameters	<p><i>iMode</i> int Determines the function of the Port:</p> <p>HTTRANSMIT_OFF Transmitter is turned off</p> <p>HTTRANSMIT_STD Transmitter transmits standard packets</p> <p>HTTRANSMIT_COL Transmitter transmits collision packets</p> <p>All other values are invalid and have no effect on the SmartBits.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range</i>: 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p> <p>NOTE: If any of iHub, iSlot, iPort are equal to –1, then the last selected port will be disabled.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	This function assumes that at least one SmartBits is attached to the SmartBits. It will be ignored by the SmartBits if there is not another SmartBits present.

HTSendCommand

Description	<p>This function is used to save a small length of time by storing up commands on the SMB, and then sending them to the cards all at once.</p> <p>This function works in conjunction with the HTSeparateHubCommands. The default setting used by HTSendCommand is HUB_DEFAULT_ACTION.</p>
Syntax	int HTSendCommand(int State)
Parameters	<p><i>State</i> int If zero, all commands that can be queued up are queued up.</p> <p>If non-zero, commands are not deferred; they are sent to the cards as soon as they reach the SMB controller. Any commands that have been deferred are sent first.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	<p>It is strongly advised that this function not be called unless the time necessary to handle each function separately is intolerably long.</p> <p>Call HTSendCommand at the beginning of a long series of commands with “State” set to 0. This causes the SMB to start buffering certain commands instead of forwarding them separately to the cards. Before the last command, call this function once again with “State” set to 1. This causes the SMB to then send all the deferred commands to the cards at once, shotgun style.</p> <p>NOTE: This function is not useful if commands are sent to the SMB across a network that adds more time than the time from controller to card.</p>

HTSeparateHubCommands

Description	Determines how commands are synchronized across multiple hubs, including whether GPS is used or not. Used in conjunction with HGRun, HGStart, HGStop, HGStep, and HTSendCommand.
Syntax	int HTSeparateHubCommands (int iFlag)
Parameters	<p><i>iFlag</i> int This value determines how if and how SmartBits chassis are synchronized.</p> <p>HUB_GROUP_DEFAULT_ACTION Enables a group action across SmartBits hubs. and stacks (GPS is not used). This setting allows a single command for stacks of hubs linked by the expansion ports (see comment). Use this value to skip GPS sync time if GPS is available but you don't want to use it. This value is the default for HGRun, HGStart, HGStep, and HGStop.</p> <p>HUB_GROUP_INDEPENDENT_ACTION Enables a group start for each SmartBits hub. No synchronization BETWEEN hubs. This setting causes a separate command to be sent for each SmartBits hub regardless of whether there are stacks, expansion connection, or GPS. This parameter was originally used to deal with older equipment that could not perform a group start across hubs.</p> <p>HUB_GROUP_SYNC_ACTION Enables GPS capability for a synchronized group start across multiple hubs. This setting allows a single command for stacks of hubs linked by the expansion ports (see comment). ERROR CONDITIONS: 1 - GPS enabled on a "Slave stack" (expansion cable plugged in the IN port.) 2 - One or more active "Links" (direct to the PC)with neither expansion con nor GPS.</p>
Return Value	The return value shows the current value set: 0 = HUB_GROUP_DEFAULT_ACTION 1 = HUB_GROUP_INDEPENDENT_ACTION 2 = <i>Reserved</i> 3 = HUB_GROUP_SYNC_ACTION
Comments	Expansion ports are available on the SMB 2000 or later. They are used to link one stack of chassis to another.

HTSetCommand

Description	Sends a command to a card which accepts HTSetCommand() actions. The commands, defines, and structure definitions for this command, for all card families, can be found in the <i>Message Functions</i> manual, which covers all the card families that allow control using HTSetCommand(), HTSetStructure(), and HTGetStructure(). The correct combination of iType parameter values and the structure parameter enable the cards to be set up in an elegant, efficient manner.
Syntax	int HTSetCommand(int iType1,int iType2,int iType3,int iType4,void* pData,int iHub, int iSlot, int iPort);
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See <i>Appendix B</i> for error codes.
Comments	See the <i>Message functions</i> manual for appropriate values for the iType and structure parameters for HTSetCommand(), HTSetStructure(), and HTGetStructure().

HTDefaultStructure

Description	Puts appropriate default values in the specified Message Function structure.
Syntax	int HTSetStructure(int iType1,void* pData,int iSize,int iHub, int iSlot, int iPort);
Parameters	<p><i>iType1</i> int defines the command action.</p> <p><i>pData</i> void* Pointer to a structure (or an array of structures) that will receive the default values.</p> <p><i>iSize</i> int indicates the size of pData. While in most cases this will be the size of the structure, in some cases it is the size of an array of structures or bytes. See the <i>Message Functions</i> manual for clarification.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See <i>Appendix B</i> for error codes.
Comments	This command is used in conjunction with the Message Functions HTSetCommand() and HTSetStructure(). Select the desired iType1 and related structure from this manual. To use this command, the defaults file (either factory-defined or customer-defined) must be either local or in your Windows directory (for MS Windows systems) or home directory (for UNIX systems). The factory-defined defaults file is named smartlib.dft .

HTSetSpeed

Description	Sets the addressed port to the selected speed. The speed selected must be appropriate to the addressed card type.
Syntax	int HTSetSpeed(int iSpeed, int iHub, int iSlot, int iPort)
Parameters	<p><i>iSpeed</i> int Determines the speed of the Port:</p> <p> SPEED_10MHZ Sets a 10MB capable card to a 10 MHZ Signaling rate (Ethernet)</p> <p> SPEED_100MHZ Sets a 100MB capable card to a 100 MHZ Signaling rate (Ethernet)</p> <p> SPEED_4MHZ Sets a 4MB capable card to a 4 MHZ Signaling rate (Token Ring)</p> <p> SPEED_16MHZ Sets a 16MB capable card to a 16 MHZ Signaling rate (Token Ring)</p> <p>All other values are invalid and have no effect on the SmartBits.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	On 100Mbps Ethernet cards, speed auto-negotiation can be enabled by configuring the MII registers. See the HTWriteMII() command for more information.

HTSetStructure

Description	Sends a command to a card which accepts HTSetStructure() actions. The commands, defines, and structure definitions for this command can be found in the <i>Message functions</i> manual, which covers all the card families that allow control using HTSetCommand(), HTSetStructure(), and HTGetStructure(). The correct combination of iType parameter values and the structure parameter enable the cards to be set up in an elegant manner.																		
Syntax	int HTSetStructure(int iType1,int iType2,int iType3,int iType4,void* pData,int iSize,int iHub, int iSlot, int iPort);																		
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"><i>iType1</i></td> <td>int defines the command action. The value (and action) depends on the card being addressed.</td> </tr> <tr> <td><i>iType2</i></td> <td>int value depends on card.</td> </tr> <tr> <td><i>iType3</i></td> <td>int value depends on card.</td> </tr> <tr> <td><i>iType4</i></td> <td>int value depends on card.</td> </tr> <tr> <td><i>pData</i></td> <td>void* Pointer to a structure or an array containing the data to send.</td> </tr> <tr> <td><i>iSize</i></td> <td>int indicates the size of the pData pointer which should be utilized. While in most cases this will be the size of the structure, in some cases it is the size of an array of structures or bytes. See the <i>Message Functions</i> manual for clarification.</td> </tr> <tr> <td><i>iHub</i></td> <td>int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</td> </tr> <tr> <td><i>iSlot</i></td> <td>int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in Hub) to 19 (last card in Hub).</td> </tr> <tr> <td><i>iPort</i></td> <td>int Identifies the port on the card or module.</td> </tr> </table>	<i>iType1</i>	int defines the command action. The value (and action) depends on the card being addressed.	<i>iType2</i>	int value depends on card.	<i>iType3</i>	int value depends on card.	<i>iType4</i>	int value depends on card.	<i>pData</i>	void* Pointer to a structure or an array containing the data to send.	<i>iSize</i>	int indicates the size of the pData pointer which should be utilized. While in most cases this will be the size of the structure, in some cases it is the size of an array of structures or bytes. See the <i>Message Functions</i> manual for clarification.	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in Hub) to 19 (last card in Hub).	<i>iPort</i>	int Identifies the port on the card or module.
<i>iType1</i>	int defines the command action. The value (and action) depends on the card being addressed.																		
<i>iType2</i>	int value depends on card.																		
<i>iType3</i>	int value depends on card.																		
<i>iType4</i>	int value depends on card.																		
<i>pData</i>	void* Pointer to a structure or an array containing the data to send.																		
<i>iSize</i>	int indicates the size of the pData pointer which should be utilized. While in most cases this will be the size of the structure, in some cases it is the size of an array of structures or bytes. See the <i>Message Functions</i> manual for clarification.																		
<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.																		
<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in Hub) to 19 (last card in Hub).																		
<i>iPort</i>	int Identifies the port on the card or module.																		
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See <i>Appendix B</i> for error codes.																		
Comments	See the <i>Message Functions</i> manual for appropriate values for the iType and structure parameters for HTSetCommand(), HTSetStructure(), and HTGetStructure().																		

HTSetTokenRingAdvancedControl

Description	Generates specialized frames for the selected TokenRing SmartCard.
Syntax	int HTSetTokenRingAdvancedControl(TokenRingAdvancedStructure *pTRAdvancedStructure, int iHub, int iSlot, int iPort)
Parameters	<p><i>PTRAdvancedStructure</i> TokenRingAdvancedStructure* Points to a TokenRingAdvancedStructure structure which contains all the information required to transmit special control frames. See the definition of TokenRingAdvancedStructure below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	This command will cause ring operation to fail if not used with caution.

TokenRingAdvancedStructure

Parameter	Description
int UseHoldingGap	Token holding gap control. 1 Activate advanced gap control. 0 Do not issue advanced gap control.
int GapValue	Time between frames when the token is not released between frames. <i>Range:</i> 1 to 1,600,000 (number of 100 nanosecond periods between frames). <i>Default value:</i> 1.
int GapScale	Scale value. NANO_SCALE Scale in nanoseconds MICRO_SCALE Scale in microseconds MILLI_SCALE Scale in milliseconds
int UseIntermediateFrameBits	Sets the Intermediate frame bit in the EDEL field of the frame. This bit is defined in the Token Ring Specification to indicate that another frame is to follow immediately, with no token being released between the frames. (See the Token Ring Architectural Specification.) 1 Set Intermediate frame 0 Clear Intermediate frame
int UseAC	Activates a user-specified Access Control field in transmitted frames. 1 Set AC from ACdata field 0 Set AC from captured token
int ACdata	Access Control byte value. Consult the <i>Token Ring Architectural Specification</i> for information on the bit fields in this byte. This byte is used to distinguish between tokens and frames and to operate the Token Priority Protocol. Setting bits in this byte incorrectly will probably cause ring errors.

int AdvancedControl1	<p>Advanced control byte 1. This byte gives the user control over how the card connects to the ring on startup and how it responds to ring errors.</p> <p>Bit 3-2: Controls connection on startup.</p> <ul style="list-style-type: none"> 0 No effect (previous settings in NVRAM are used). 1 Connects to the ring on startup (default). 2 Stays off the ring on startup. 3 Stays off the ring on startup and allows bit 1 to control the connection. <p>Bit 1: Connection control</p> <ul style="list-style-type: none"> 0 Deinserted 1 Inserted <p>Bit 0: 'Halt on Error' Stops card from transmitting when it receives a Beacon, Claim or Purge frame.</p> <ul style="list-style-type: none"> 0 Inactive 1 Active
int AdvancedControl2	<p>Advanced control byte 2.</p> <p>Bit 4: Internal Loopback</p> <ul style="list-style-type: none"> 0 Off 1 On <p>Bit 3: Test Mode. This mode is used to simulate an Active Monitor when running as a Station so that the card can be used standalone to test passive Token Ring components.</p> <ul style="list-style-type: none"> 0 Off 1 On
unsigned long AReserved1	<i>Reserved.</i>
unsigned long AReserved2	<i>Reserved.</i>

HTSetTokenRingErrors

Description	Generates traffic with error frames for the selected TokenRing SmartCard.												
Syntax	int HTSetTokenRingErrors(int ErrorTrafficRatio, int iTRErrors, int iHub, int iSlot, int iPort)												
Parameters	<p><i>ErrorTrafficRatio</i> int Specifies the error traffic ratio in tenths of seconds. <i>Range:</i> 0 to 1000. A value of 0 will turn off error generation.</p> <p><i>iTRErrors</i> int Specifies the type of frame errors to generate. Value can be a combined OR of the following defines:</p> <table style="margin-left: 20px;"> <tr> <td>TR_ERR_FCS</td> <td>FCS errors</td> </tr> <tr> <td>TR_ERR_FRAME_COPY</td> <td>Frame copy errors</td> </tr> <tr> <td>TR_ERR_FRAME_BIT</td> <td>Frame Bit errors</td> </tr> <tr> <td>TR_ERR_FRAME_FS</td> <td>FS Frame errors</td> </tr> <tr> <td>TR_ERR_ABORT_DELIMITER</td> <td>Abort delimiter errors</td> </tr> <tr> <td>TR_ERR_BURST</td> <td>Burst errors</td> </tr> </table> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>	TR_ERR_FCS	FCS errors	TR_ERR_FRAME_COPY	Frame copy errors	TR_ERR_FRAME_BIT	Frame Bit errors	TR_ERR_FRAME_FS	FS Frame errors	TR_ERR_ABORT_DELIMITER	Abort delimiter errors	TR_ERR_BURST	Burst errors
TR_ERR_FCS	FCS errors												
TR_ERR_FRAME_COPY	Frame copy errors												
TR_ERR_FRAME_BIT	Frame Bit errors												
TR_ERR_FRAME_FS	FS Frame errors												
TR_ERR_ABORT_DELIMITER	Abort delimiter errors												
TR_ERR_BURST	Burst errors												
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .												
Comments	The number in the ratio is nominally in tenths of a percent. However, as it is rationalized to a count the precision will be poor at large percentage values.												

HTSetTokenRingLLC

Description	Configures LLC frame for the selected TokenRing SmartCard.
Syntax	int HTSetTokenRingLLC(TokenRingLLCStructure *pTRLStructure, int iHub, int iSlot, int iPort)
Parameters	<p><i>pTRLStructure</i> TokenRingLLCStructure* Points to a TokenRingLLCStructure (see definition below) which contains all the information required to preform LLC Type 1 frames. See the definition of TokenRingLLCStructure below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	A TokenRing MAC header has to be defined first for LLC to take effect.

TokenRingLLCStructure

Parameter	Description
int UseLLC	Logical Link Control (LLC). 0 No LLC added to MAC frame header. 1 Add LLC to the MAC frame header.
int DSAP	Destination Service Access Point. Range: 0 to 255 (0x00 to 0xFF).
int SSAP	Source Service Access Point. Range: 0 to 255 (0x00 to 0xFF).
int LLCCommand	Sets the type of LLC field to be added to the frame header. 0 TEST frame set to 'Poll' 1 SNAP frame (used to encapsulate an Ethernet frame from the 'type' field)

HTSetTokenRingMAC

Description	Configures TokenRing MAC header for the selected TokenRing SmartCard.
Syntax	int HTSetTokenRingMAC(TokenRingMACStructure *pTRMStructure, int iHub, int iSlot, int iPort)
Parameters	<p><i>pTRMStructure</i> TokenRingMACStructure* Points to a TokenRingMACStructure which defines a preformed MAC header. See the definition of TokenRingMACStructure below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

TokenRingMACStructure

Parameter	Description
int UseMAC	MAC header control. The MAC header consists of AC and FC bytes, followed by MAC destination and source addresses, followed by optional LLC control, followed by optional SourceRouteAddress information. AC and FC are always prepended to frame data. 0 No MAC header prepended to frame data. 1 Prepend a MAC header to the frame data.
int Stations	<i>Reserved</i> . Must be set to 1.
int MACSrc[6]	Source MAC Address.
int MACDest[6]	Destination MAC Address.
int FramesPerToken	Number of frames to be transmitted for each token. <i>Range:</i> 1 to 340 (0x01 to 0x154).

int FrameControl	<p>This is the value of the Frame Control byte put on the front of each frame. This byte is independent of the fill pattern and any preformed header may be overwritten by a VFD field.</p> <p>This byte is defined fully in the <i>Token Ring Architectural Specification</i> and should not be altered from the default value of 0x40 (TRFC_DEFAULT) without knowledge of the consequences.</p> <p>Several other values are defined in the header file:</p> <table border="0"> <tr> <td>TRFC_DEFAULT</td> <td>Standard frame</td> </tr> <tr> <td>TRFC_PCF_BEACON</td> <td>Beacon</td> </tr> <tr> <td>TRFC_PCF_CLAIMTOKEN</td> <td>Claim Token</td> </tr> <tr> <td>TRFC_PCF_RINGPURGE</td> <td>Ring Purge</td> </tr> <tr> <td>TRFC_PCF_AMP</td> <td>Active Monitor Present</td> </tr> <tr> <td>TRFC_PCF_SMP</td> <td>Standby Monitor Present</td> </tr> <tr> <td>TRFC_PCF_DAT</td> <td>Duplicate Address Test</td> </tr> <tr> <td>TRFC_PCF_RRS</td> <td>Remove Ring Station</td> </tr> </table>	TRFC_DEFAULT	Standard frame	TRFC_PCF_BEACON	Beacon	TRFC_PCF_CLAIMTOKEN	Claim Token	TRFC_PCF_RINGPURGE	Ring Purge	TRFC_PCF_AMP	Active Monitor Present	TRFC_PCF_SMP	Standby Monitor Present	TRFC_PCF_DAT	Duplicate Address Test	TRFC_PCF_RRS	Remove Ring Station
TRFC_DEFAULT	Standard frame																
TRFC_PCF_BEACON	Beacon																
TRFC_PCF_CLAIMTOKEN	Claim Token																
TRFC_PCF_RINGPURGE	Ring Purge																
TRFC_PCF_AMP	Active Monitor Present																
TRFC_PCF_SMP	Standby Monitor Present																
TRFC_PCF_DAT	Duplicate Address Test																
TRFC_PCF_RRS	Remove Ring Station																

HTSetTokenRingProperty

Description	Configures ring operation characteristics for the selected TokenRing SmartCard.
Syntax	int HTSetTokenRingProperty(TokenRingPropertyStructure *pTRPStructure, int iHub, int iSlot, int iPort)
Parameters	<p><i>pTRPStructure</i> TokenRingPropertyStructure* Points to a TokenRingPropertyStructure (see definition below) which contains all the information required to configure ring operation.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	This command defines card properties retained in non-volatile storage. Do not alter these parameters on a live ring; this can cause ring malfunction (usually Beaconsing by other stations which might cause them to close down pending a hard reset).

TokenRingPropertyStructure

Parameter	Description
int SpeedSetting	Ring speed. TR_SPEED_4MBITS 4 Mbits/Sec TR_SPEED_16MBITS 16 Mbits/Sec
int EarlyTokenRelease	Allows a station to transmit a token immediately after a frame was sent. This feature only applies to a ring running at 16 Mbps. TR_TOKEN_DEFAULT Do not allow TR_TOKEN_EARLY_RELEASE Allow
int DuplexMode	Half duplex or full duplex. TR_DUPLEX_HALF TKP Half duplex TR_DUPLEX_FULL TXI Full duplex
int DeviceOrMAUMode	Configures the TokenRing SmartCard to be a port or a station. TR_MODE_MAU Port TR_MODE_DEVICE Station

HTSetTokenRingSrcRouteAddr

Description	Configures Source Route Address (SRA) for the selected TokenRing SmartCard.	
Syntax	int HTSetTokenRingSrcRouteAddr(int UseSRA, int *piData, int iHub, int iSlot, int iPort)	
Parameters	<i>UseSRA</i>	int specifies if a SRA field is included in a TokenRing frame. 0 No SRA defined 1 Use SRA field defined in piData parameter.
	<i>piData</i>	int * Points to an array of int which contains the Source Route Address information. The maximum length of this array is 32 and the length information is encoded in the lower 5 bits of the first byte of this array of SourceRouteAddress information.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	This field is part of a pre-formed header and so the MAC header has to be active for it to be active. The data in this field will be parsed by the card to determine the size of the source routing field to use and the maximum frame size to transmit. (See the <i>Token Ring Architectural Reference</i> for details of how to code this field.)	

HTSetVGProperty

Description	Configures ring operation characteristics for the selected VG SmartCard.	
Syntax	int HTSetVGProperty(VGCardPropertyStructure *pVGPStructure)	
Parameters	<i>pVGPStructure</i>	VGCardPropertyStructure* Points to a VGCardPropertyStructure (see definition below) which contains all the information required to configure Card.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.
	<i>iSlot</i>	int Identifies the slot where the TokenRing SmartCard is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the port on the card or module.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	None	

VGCardPropertyStructure

Parameter	Description
int EndOrMasterNode	Allows a VG SmartCard to be configured as an End node or a Master node. VG_CFG_END_NODE End Node VG_CFG_MASTER Master Node
int PriorityPromotion	Priority promotion. VG_CFG_NO_PRIO_PROMO No promotion VG_CFG_PRIORITY_PROMO Yes
int EtherNetOrTokenRing	Configures the VG SmartCard to be operated in Ethernet or in TokenRing. VG_CFG_ETHERNET Ethernet CG_CFG_TOKENRING TokenRing

HTSlotOwnership

Description	Identifies the ownership status of a specified slot in a multi-user environment.
Syntax	int HTSlotOwnership(int iHub, int iSlot)
Parameters	<p><i>iHub</i> int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs-1). Hub IDs start at 0.</p> <p><i>iSlot</i> int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through last slot.</p>
Return Value	<p>If the function executed successfully, the return value shows the current owner of the specified slot.</p> <p>0 SLOT_RESERVED_BY_OTHER</p> <p>1 SLOT_RESERVED_BY_USER</p> <p>2 SLOT_AVAILABLE</p> <p>A failure code (less than zero) is returned if the function failed. <i>See Appendix B for error codes.</i></p>
Comments	<p>This command may be used only with SmartBits 6000/600 chassis.</p> <p>Use this function to determine the current reservation state of the specified slot. It will return different values depending on whether the slot is reserved by the current user, reserved by a different user, present but not reserved, or not present.</p>

HTSlotRelease

Description	Releases a slot from use in a multi-user environment, making it available for reservation by other users.				
Syntax	int HTSlotReserve(int iHub, int iSlot)				
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><i>iHub</i></td> <td>int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs–1). Hub IDs start at 0.</td> </tr> <tr> <td><i>iSlot</i></td> <td>int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through last slot.</td> </tr> </table>	<i>iHub</i>	int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs–1). Hub IDs start at 0.	<i>iSlot</i>	int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through last slot.
<i>iHub</i>	int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs–1). Hub IDs start at 0.				
<i>iSlot</i>	int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through last slot.				
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code (less than zero) is returned if the function failed. <i>See Appendix B for error codes.</i>				
Comments	<p>This command may be used only with SmartBits 6000/600 chassis.</p> <p>This function releases a slot, marking it as free to be reserved using the HTSlotReserve function.</p> <p>Releasing a card (slot) has no effect on card configuration.</p> <p>Releasing a card that the user does not already have reserved is legal and will return a code indicating success.</p>				

HTSlotReserve

Description	Marks the named slot as reserving exclusive read/write access to the issuing user, in a multi-user environment.				
Syntax	int HTSlotReserve(int iHub, int iSlot)				
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><i>iHub</i></td> <td>int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs–1). Hub IDs start at 0.</td> </tr> <tr> <td><i>iSlot</i></td> <td>int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through 5 (last slot).</td> </tr> </table>	<i>iHub</i>	int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs–1). Hub IDs start at 0.	<i>iSlot</i>	int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through 5 (last slot).
<i>iHub</i>	int Identifies the hub where the card or module is installed. <i>Range:</i> 0 (first hub) through N (number of hubs–1). Hub IDs start at 0.				
<i>iSlot</i>	int Identifies the slot where the card or module is installed. <i>Range:</i> 0 (first slot) through 5 (last slot).				
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code (less than zero) is returned if the function failed. <i>See Appendix B for error codes.</i>				
Comments	<p>This command may be used only with SmartBits 6000/600 chassis.</p> <p>This function reserves a slot (which may contain more than one port) for exclusive use by the link (user) on which the command was invoked. Once the slot has been reserved, no other user may reserve the slot until it is released, either by the reserving user or the super user.</p> <p>The controller cannot know whether or not multiple TCP/IP links come from the same library instance. For this reason, reservations to a controller on one link are not accessible from the same library instance using a second link to the same controller.</p> <p>Reserving a card (slot) has no effect on card configuration.</p> <p>Reserving a card that the user has already reserved is legal and will return a code indicating success.</p>				

HTSymbol

Description	Generates symbol error for the 100 Mbps card. The port can be set up to transmit a series of packets that generates an invalid wave form data pattern. This command applies only to 100 Mbps cards.
Syntax	int HTSymbol(int Mode, int iHub, int iSlot, int iPort)
Parameters	<p><i>Mode</i> int Identifies the symbol mode of the board. Legal modes can be conveyed using the following constants:</p> <p>SYMBOL_OFF Turn off symbol errors</p> <p>SYMBOL_ON Turn on symbol errors</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTTransmitMode

Description	Indicates to the selected port how to control the transmission of packets when running.
Syntax	int HTTransmitMode(int iMode, int iHub, int iSlot, int iPort)
Parameters	<p><i>iMode</i> int Indicates the mode of operation when transmitting packets according to the following defines:</p> <p>CONTINUOUS_PACKET_MODE Sets port to transmit single packets continuously.</p> <p>SINGLE_BURST_MODE Sets port to transmit a single burst of packets, then stop.</p> <p>MULTI_BURST_MODE Sets port to transmit multiple bursts of packets, indicated by the HxMultiBurstCount command, with each burst being separated by the amount specified in the HxBurstGap command, and then stop. (For the above commands, "Hx" indicates HT or HG.)</p> <p>CONTINUOUS_BURST_MODE Sets port to repetitively send bursts of packets with each burst being separated by the amount specified in the HxBurstGap command.</p> <p>ECHO_MODE Sets port to transmit a single packet upon receiving a Receive Trigger event. The packet transmitted will match the programmed parameters of the port addressed.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTTrigger

Description	Sets up the triggering mechanism for a card. HTTrigger specifies the trigger number (1 or 2), the operational configuration, trigger pattern range, trigger pattern offset, and trigger pattern data.
Syntax	int HTTrigger(int iTrigId, int iConfig, HTTriggerStructure* pHTStruct, int iHub, int iSlot, int iPort)
Parameters	<p><i>iTrigId</i> int Identifies the trigger source. There are two possible triggers on each card. They are identified as follows:</p> <p style="padding-left: 40px;">HTTRIGGER_1 Trigger 1 HTTRIGGER_2 Trigger 2</p> <p><i>iConfig</i> int There are three possible types of configurations for the triggers on the cards:</p> <p style="padding-left: 40px;">HTTRIGGER_OFF Disables the triggering mechanism for TrigId HTTRIGGER_ON Enables the triggering mechanism for TrigId HTTRIGGER_DEPENDENT Enables the triggering mechanism for TrigId after the other trigger has triggered.</p> <p><i>pHTStruct</i> HTTriggerStructure* A structure containing the trigger pattern, offsets and ranges. Note that the maximum range is 6 bytes, and, though the range is specified in bits., the specified number is rounded up to the nearest byte multiple. i.e.; the cards can only trigger on patterns that are a length that is a multiple of 8 bits. The offset ranges from 1 to 12,112 bits (specified in bits). See the definition of <i>HTTriggerStructure</i> on page 141.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	<p>It is possible to misconfigure triggers when using HTTRIGGER_DEPENDENT. A TrigId set to HTTRIGGER_DEPENDENT is to be active after the other TrigId trigger has occurred. So, if trigger 2 is set to be dependent on trigger 1:</p> <p>A properly configured trigger dependent combination would be (the order of the commands does matter):</p> <pre style="padding-left: 40px;">HTTrigger (HTTRIGGER_1, HTTRIGGER_ON, &TStruct, 0, 0, 1) HTTrigger (HTTRIGGER_2, HTTRIGGER_DEPENDENT, &TStruct, 0, 0, 1)</pre> <p>A misconfigured trigger combination would be:</p> <pre style="padding-left: 40px;">HTTrigger (HTTRIGGER_1, HTTRIGGER_OFF, &TStruct, 0, 0, 1) HTTrigger (HTTRIGGER_2, HTTRIGGER_DEPENDENT, &TStruct, 0, 0, 1)</pre> <p>Here, trigger 2 will never fire because trigger 1 is off.</p>

HTVFD

Description	Sends VFD information to a card.							
Syntax	int HTVFD(int iVFDId, HTVFDStructure* phtHStruct,int iHub, int iSlot, int iPort)							
Parameters	<p><i>iVFDId</i> int Identifies the VFD pattern being addressed. There are a total of three VFD patterns. They are identified as shown below:</p> <table style="margin-left: 40px;"> <tr> <td>HVFD_1</td> <td>VFD Pattern 1</td> </tr> <tr> <td>HVFD_2</td> <td>VFD Pattern 2</td> </tr> <tr> <td>HVFD_3</td> <td>VFD Pattern 3</td> </tr> </table> <p><i>phtHStruct</i> HTVFDStructure* pointer to a structure that holds VFD information for use with a card. This structure holds the VFD Configuration, Range, Offset and Pattern. See the definition of HTVFDStructure below.</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>	HVFD_1	VFD Pattern 1	HVFD_2	VFD Pattern 2	HVFD_3	VFD Pattern 3	
HVFD_1	VFD Pattern 1							
HVFD_2	VFD Pattern 2							
HVFD_3	VFD Pattern 3							
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.							
Comments	None							

HTVFDStructure

Parameter	Description														
int Configuration	<p>Determines the capabilities of the VFD being implemented. Select the constant that applies.</p> <p>Configurations specific to VFD1 and VFD2 are:</p> <table style="margin-left: 40px;"> <tr> <td>HVFD_NONE</td> <td>VFD off</td> </tr> <tr> <td>HVFD_RANDOM</td> <td>Random pattern</td> </tr> <tr> <td>HVFD_INCR</td> <td>Incrementing pattern</td> </tr> <tr> <td>HVFD_DECR</td> <td>Decrementing pattern</td> </tr> <tr> <td>HVFD_STATIC</td> <td>Static pattern</td> </tr> </table> <p>Configuration options for VFD3 are:</p> <table style="margin-left: 40px;"> <tr> <td>HVFD_NONE</td> <td>VFD3 off</td> </tr> <tr> <td>HVFD_ENABLED</td> <td>VFD3 on</td> </tr> </table> <p>NOTE: VFD3 operates differently from 1 and 2. It is a large buffer that can be used in segments to create more complex patterns than increment or decrement.</p>	HVFD_NONE	VFD off	HVFD_RANDOM	Random pattern	HVFD_INCR	Incrementing pattern	HVFD_DECR	Decrementing pattern	HVFD_STATIC	Static pattern	HVFD_NONE	VFD3 off	HVFD_ENABLED	VFD3 on
HVFD_NONE	VFD off														
HVFD_RANDOM	Random pattern														
HVFD_INCR	Incrementing pattern														
HVFD_DECR	Decrementing pattern														
HVFD_STATIC	Static pattern														
HVFD_NONE	VFD3 off														
HVFD_ENABLED	VFD3 on														
int Range	<p>Determines the length of the VFD field that will be laid into the frame.</p> <p>For VFD1 and VFD2:</p> <p>To specify the length in byte units, use a positive integer from 1 to 6. To specify the length in bit units, use a negative integer from –1 to –48. The minus symbol flags the library that the number represents bits instead of bytes. Since 100Mbps Ethernet cards send traffic in increments of four bits, a range that is not in multiples of four will be rounded up to the nearest nibble for these cards.</p>														

Parameter	Description
	<p>For VFD3: The length of VFD3 is set in bytes. The byte length is from 1 to 2047.</p>
int Offset	<p>Determines the bit number in the frame where VFD is overlaid. Measurement begins immediately after the preamble. <i>Range</i>: 0 to 12,112. For a 100Mbps Ethernet SmartCard, values that are not multiples of four are rounded up to the next 4 bit (nibble) increment.</p>
int Data	<p>Points to an array of integers that constitute the pattern for the VFD. For Visual Basic, use int*iData instead of int*Data. For VFD1 and VFD2 only: Elements values are entered into the array with the most significant bit first. For example: iData[0] 0 iData[1] 1 iData[2] 2 iData[3] 3 iData[4] 4 iData[5] 5 Creates the VFD pattern: 543210[BS18]</p>
int DataCount	<p>This value has different uses for VFD1 or 2 and for VFD3. For VFD1 and VFD2: The DataCount is used with Configuration to limit the number of patterns generated. DataCount is the Cycle-count (number of different patterns that will be generated before being repeated). Example 1: If Configuration = HVFD_INCR And if DataCount = 6 Results in six VFD patterns. The initial pattern is used in the first frame. The next five values increment, creating a series of five new patterns. The initial pattern is then used again, and the cycle repeats itself. Example 2: If Configuration = HVFD_INCR And if DataCount = 0 The VFD increments the full value that the Range allows, and then cycles over again. For VFD3: The buffer size of the Data array. Used in combination with the Range to determine how often a pattern is repeated. For example, if the DataCount is 24 and the Range is 6, there will be four six byte patterns before the first is repeated. For Gigabit Ethernet cards, the byte length is from 1 to 16384. For all other SmartCards, the byte length is from 1 to 2047[BS19].</p>

HTWriteMII

Description	Writes a specific MII Address/Register. This command applies only to 100 Mbps cards.
Syntax	int HTWriteMII(unsigned int uiAddress, unsigned int uiRegister, unsigned short uiBits, int iHub, int iSlot, int iPort)
Parameters	<p><i>uiAddress</i> unsigned int Specific address. Must be from 0 to 31.</p> <p><i>uiRegister</i> unsigned int Specific register. Must be from 0 to 31.</p> <p><i>uiBits</i> unsigned short Bit value to write to address/register.</p> <p><i>iHub</i> int Identifies the hub where the is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Hub Numbering with Multiple Chassis</i> in Chapter 2.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the port on the card or module.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B</i> for error codes.
Comments	None

NSCreateFrame

Description	Automates and simplifies creation of frames with the use of the structure: <code>FrameSpec_Type</code> .
Syntax	long NSCreateFrame(<code>FrameSpec_Type*</code> framespec)
Parameters	<p><i>framespec</i> FrameSpec_Type* pointer to a structure that holds information about the type of frame(s) to be created. Elements shown below can have a wide variety of values. For values of <i>iEncap</i>, <i>iSize</i>, <i>iProtocol</i>, and <i>iPattern</i>, see the definition of FrameSpec_Type below.</p> <p><i>iEncap</i> The type of frame (Ethernet, ATM, etc.)</p> <p><i>iSize</i> The size of the frame. If <i>iSize</i> value is either too large or too small (based on selected <i>iEncap</i> and <i>iProtocol</i> values), an error value is returned.</p> <p><i>iProtocol</i> The type of Layer 3 protocol to use, e.g., IP, ARP, None, etc.</p> <p><i>iPattern</i> The background pattern to use. This pattern is used to pad the frame (to match the <i>iSize</i> value) after all specified bytes have been inserted.</p>
Return Value	If successful, a Frame ID is returned. This ID represents a single frame prototype. Use this ID to put the frame in the card buffer with <code>HTFrame</code> . If failure occurs, a negative integer is returned. See <i>Appendix B</i> for error codes.
Comments	<p>For a custom payload (background pattern), set the <i>iPattern</i> to <code>PAT_CUST</code>, and then create the custom pattern with NSSetPayload.</p> <p>Once a frame is created, put it into the card transmit buffer using the HTFrame function. (This function is similar to <code>HTFillPattern</code>.)</p> <p>Related functions: <code>NSDeleteFrame</code>, <code>NSCreateFrameAndPayload</code>, and <code>NSModifyFrame</code>.</p> <p>Since <code>NSCreateFrame</code> functions are intended for “layer 2” mode, VTEs and Signature fields are not part of these frames.</p>

FrameSpec_Type

This structure is used with NSCreateFrame and NSCreateFrameAndPayload.

Parameter	Description
int iEncap	Type of frame encapsulation used. In addition to iEncap, this information determines the value of the iSize variable. ENCAP_ETHERNET ENCAP_ATM_PVC ENCAP_ATM_SVC_SNAP ENCAP_ATM_SVC_LANE802_3 ENCAP_ATM_SVC_LANE802_5 ENCAP_ATM_SVC_CLASSICAL_IP ENCAP_TOKEN_RING ENCAP_BRIDGE_FR Frame Relay ENCAP_ROUTE_FR Frame Relay
int iSize	Specifies the size of the frame prototype being created. The maximum size is 2K bytes. Set the frame size to be large enough to contain the encapsulation information and protocol header. Any extra space left over will be filled by the iPattern value. CRC and Preamble are not included in this frame size. An example of frame size is: (Encapsulation w/ 2 bytes for protocol added once protocol is selected) + (protocol) +(optional payload bytes)
int iProtocol	Specifies what type of protocol header is used. In addition to iEncap, this information determines the value of the iSize variable. FRAME_PROTOCOL_NULL No protocol header used. The background-fill pattern pads the frame after the encapsulation bytes. FRAME_PROTOCOL_IP FRAME_PROTOCOL_UDP FRAME_PROTOCOL_TCP FRAME_PROTOCOL_ARP FRAME_PROTOCOL_RARP FRAME_PROTOCOL_IPX FRAME_PROTOCOL_ICMP

NSCreateFrameAndPayload

Description	Automates and simplifies creation of frames with the use of the structure: <code>FrameSpec_Type</code> . For use <i>only</i> with a customized payload (fill pattern). For predefined SmartLib payload, use <code>NSCreateFrame</code> .														
Syntax	<code>long NSCreateFrameAndPayload(FrameSpec_Type* framespec, int iPayloadSize, unsigned char* pucPayload)</code>														
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"><i>framespec</i></td> <td>FrameSpec_Type* pointer to a structure that holds information about the type of frame(s) to be created. <i>Structure elements</i> shown below can have a wide variety of values. For values of <code>iEncap</code>, <code>iSize</code>, <code>iProtocol</code>, and <code>iPattern</code>, see the definition of FrameSpec_Type above.</td> </tr> <tr> <td><code>iEncap</code></td> <td>The type of frame (Ethernet, ATM, etc.)</td> </tr> <tr> <td><code>iSize</code></td> <td>The size of the frame. If <code>iSize</code> value is either too large or too small (based on selected <code>iEncap</code> and <code>iProtocol</code> values), an error value is returned.</td> </tr> <tr> <td><code>iProtocol</code></td> <td>The type of Layer 3 protocol to use, e.g., IP, ARP, None, etc.</td> </tr> <tr> <td><code>iPattern</code></td> <td>The background pattern to use. For this function the only valid value is: <code>PAT_CUST</code>.</td> </tr> <tr> <td><i>iPayloadSize</i></td> <td>int Specifies the length of the payload (fill pattern) array.</td> </tr> <tr> <td><i>pucPayload</i></td> <td>unsigned char Pointer to user-created array containing the customized payload (fill pattern).</td> </tr> </table>	<i>framespec</i>	FrameSpec_Type* pointer to a structure that holds information about the type of frame(s) to be created. <i>Structure elements</i> shown below can have a wide variety of values. For values of <code>iEncap</code> , <code>iSize</code> , <code>iProtocol</code> , and <code>iPattern</code> , see the definition of FrameSpec_Type above.	<code>iEncap</code>	The type of frame (Ethernet, ATM, etc.)	<code>iSize</code>	The size of the frame. If <code>iSize</code> value is either too large or too small (based on selected <code>iEncap</code> and <code>iProtocol</code> values), an error value is returned.	<code>iProtocol</code>	The type of Layer 3 protocol to use, e.g., IP, ARP, None, etc.	<code>iPattern</code>	The background pattern to use. For this function the only valid value is: <code>PAT_CUST</code> .	<i>iPayloadSize</i>	int Specifies the length of the payload (fill pattern) array.	<i>pucPayload</i>	unsigned char Pointer to user-created array containing the customized payload (fill pattern).
<i>framespec</i>	FrameSpec_Type* pointer to a structure that holds information about the type of frame(s) to be created. <i>Structure elements</i> shown below can have a wide variety of values. For values of <code>iEncap</code> , <code>iSize</code> , <code>iProtocol</code> , and <code>iPattern</code> , see the definition of FrameSpec_Type above.														
<code>iEncap</code>	The type of frame (Ethernet, ATM, etc.)														
<code>iSize</code>	The size of the frame. If <code>iSize</code> value is either too large or too small (based on selected <code>iEncap</code> and <code>iProtocol</code> values), an error value is returned.														
<code>iProtocol</code>	The type of Layer 3 protocol to use, e.g., IP, ARP, None, etc.														
<code>iPattern</code>	The background pattern to use. For this function the only valid value is: <code>PAT_CUST</code> .														
<i>iPayloadSize</i>	int Specifies the length of the payload (fill pattern) array.														
<i>pucPayload</i>	unsigned char Pointer to user-created array containing the customized payload (fill pattern).														
Return Value	If successful, a Frame ID is returned. This ID represents a single frame prototype. Use this ID to put the frame in the card buffer using the HTFrame function. (This function is similar to <code>HTFillPattern</code> .) If failure occurs, a negative integer is returned. <i>See Appendix B for error codes.</i>														
Comments	If you want to use a pre-created fill pattern, use <code>NSCreateFrame</code> . A second way to accomplish the same task (a frame with a custom fill pattern) is to use the <code>NSCreateFrame</code> function, using <code>PAT_CUST</code> for the <code>iPattern</code> parameter, and then defining the custom pattern with <code>NSSetPayload</code> . Once a frame is created, put it into the card transmit buffer using <code>HTFrame</code> . (This function is similar to <code>HTFillPattern</code> .) Related functions: <code>NSDeleteFrame</code> and <code>NSModifyFrame</code> .														

NSDeleteFrame

Description	Deletes a single frame prototype specified by the IFrameID. The frame prototype is identified by the Frame ID (which is returned by NSCreateFrame and NSCreateFrameAndPayload).
Syntax	long NSDeleteFrame (long iFrameID)
Parameters	<i>IFrameID</i> long The ID number is unique to each frame prototype, and is returned when a frame is created. Use the iFrameID value to put the frame in the card buffer with HTFrame, and to delete the frame from the SmartLib buffer with NSDeleteFrame.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure value of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Use NSDeleteFrame to clear the Prototype From the SmartLib buffer once that type of frame is no longer needed. Related functions: HTFrame, NSCreateFrame, NSSetPayload, NSCreateFrameAndPayload, and NSModifyFrame.

NSDisableAutoDefaults

Description	(TCL INTERFACE ONLY) Turns off automated defaults so that HTSetDefault must be called to use default values for Message Function parameters.
Syntax	NSDisableAutoDefaults
Parameters	
Return Value	0
Comments	This command requires no parameters. <i>See Working with Tcl</i> in this manual for more information. See also: NSEnableAutoDefaults

NSEnableAutoDefaults

Description	(TCL INTERFACE ONLY) Used at the top of a script. Once called, default values are automatically supplied for Message Functions that require settings.
Syntax	NSEnableAutoDefaults
Parameters	
Return Value	1
Comments	This command requires no parameters. <i>See Working with Tcl</i> in this manual for more information. See also: NSDisableAutoDefaults

NSGetMaxHubs

Description	Returns the maximum number of hubs per stack. This number is 4 whether the connection is a stack of SMB 2000s or an SMB 6000.
Syntax	int NSGetMaxHubs (void)
Parameters	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Replaces the constant MAX_HUBS. This value is useful for allocating memory.

NSGetMaxPorts

Description	Returns the maximum number of ports per stack.
Syntax	int NSGetMaxPorts (void)
Parameters	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Replaces the constant MAX_PORTS. This value is useful when allocating memory.

NSGetMaxSlots

Description	Returns the maximum number of slots per stack.
Syntax	int NSGetMaxSlots (void)
Parameters	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Replaces the constant MAX_SLOTS. This value is useful when allocating memory.

NSGetNumHubs

Description	Returns the number of hubs possible for the hub type.
Syntax	int NSGetNumHubs (void)
Parameters	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	The number returned is 4 for SMB-2000, 1 for SMB-6000, 600, and 200.

NSGetNumPorts

Description	Returns the number of ports available for a given card. For SmartCards, there will be a single port per card. For SmartModules there will be multiple ports.
Syntax	int NSGetNumPorts(int iHub, int iSlot);
Parameters	<p><i>iHub</i> int Specifies which hub to query. The range is 0 (first hub) through N (the number of available hubs -1).</p> <p><i>iSlot</i> int Specifies which slot to query. The range is 0 (first hub) through N (the number of available hubs -1).</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	

NSGetNumSlots

Description	Returns the number of slots possible for the specified hub (this does Not return the number of cards available).
Syntax	int NSGetNumSlots(int iHub);
Parameters	<i>iHub</i> int Specifies which hub to query. The range is 0 (first hub) through N (the number of available hubs -1).
Return Value	The return value is >= 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	

NSModifyFrame

Description	Modifies part of a frame created by NSCreateFrame or NSCreateFrameAndPayload. It can be used for a series of frames based on an original frame prototype.																																																																																								
Syntax	Long NSModifyFrame (long IFrameID, int iIdentifier, unsigned char* pucBytes, int iNumBytes)																																																																																								
Parameters	<p><i>IFrameID</i> long The FrameID number is unique for each frame prototype. It is returned by NSCreateFrame and NSCreateFrameAndPayload.</p> <p><i>iIdentifier</i> int This value specifies which portion of the frame to modify. For example, you might modify the Destination MAC, or the Time-to-Live, etc. By selecting an element, you do not need to know it's offset, only its size and content. Some elements can modify "Only" one type of frame, while others have multiple uses defined by "All Followed."</p> <table border="0"> <tr> <td>FRAME_VERSION</td> <td>"ONLY" IP version.</td> </tr> <tr> <td>FRAME_HEADER_LENGTH</td> <td>"ALL FOLLOWED:" IP length, IPX length.</td> </tr> <tr> <td>FRAME_UDP_HEADER_LENGTH</td> <td>"ONLY" UDP length.</td> </tr> <tr> <td>FRAME_TCP_HEADER_LENGTH</td> <td>"ONLY" TCP length.</td> </tr> <tr> <td>FRAME_TYPE_SERVICE</td> <td>"ONLY" IP type of service.</td> </tr> <tr> <td>FRAME_TOTAL_LENGTH</td> <td>"ONLY" IP total length.</td> </tr> <tr> <td>FRAME_SEQUENCE</td> <td>"ALL FOLLOWED:" IP sequence, ICMP sequence.</td> </tr> <tr> <td>FRAME_UDP_SEQUENCE</td> <td>"ONLY" UDP sequence.</td> </tr> <tr> <td>FRAME_TCP_SEQUENCE</td> <td>"ONLY" TCP sequence.</td> </tr> <tr> <td>FRAME_FLAGS</td> <td>"ONLY" IP flags.</td> </tr> <tr> <td>FRAME_FRAGMENTS_OFFSET</td> <td>"ONLY" IP fragment and offset.</td> </tr> <tr> <td>FRAME_TIME_TO_LIVE</td> <td>"ONLY" IP time to live.</td> </tr> <tr> <td>FRAME_PROTOCOL</td> <td>"ONLY" IP protocol.</td> </tr> <tr> <td>FRAME_HEADER_CRC</td> <td>"ALL FOLLOWED:" IP checksum, IPX checksum.</td> </tr> <tr> <td>FRAME_UDP_HEADER_CRC</td> <td>"ONLY" UDP checksum.</td> </tr> <tr> <td>FRAME_TCP_HEADER_CRC</td> <td>"ONLY" TCP checksum.</td> </tr> <tr> <td>FRAME_DST_IP_ADDR</td> <td>"ANY" frame Destination IP Address.</td> </tr> <tr> <td>FRAME_SRC_IP_ADDR</td> <td>"ANY" frame Source IP Address.</td> </tr> <tr> <td>FRAME_SRC_PORT</td> <td>"ANY" frame Source Port number</td> </tr> <tr> <td>FRAME_DST_PORT</td> <td>"ANY" frame Destination Port number.</td> </tr> <tr> <td>FRAME_ACKNOWLEDGE</td> <td>"ONLY" TCP Acknowledge number.</td> </tr> <tr> <td>FRAME_RESERVED</td> <td>"ONLY" TCP reserved bits.</td> </tr> <tr> <td>FRAME_URG_BIT</td> <td>"ONLY" TCP URG bit.</td> </tr> <tr> <td>FRAME_ACK_BIT</td> <td>"ONLY" TCP ACK bit.</td> </tr> <tr> <td>FRAME_PSH_BIT</td> <td>"ONLY" TCP PSH bit.</td> </tr> <tr> <td>FRAME_RST_BIT</td> <td>"ONLY" TCP RST bit.</td> </tr> <tr> <td>FRAME_SYN_BIT</td> <td>"ONLY" TCP SYN bit.</td> </tr> <tr> <td>FRAME_FIN_BIT</td> <td>"ONLY" TCP FIN bit.</td> </tr> <tr> <td>FRAME_WINDOW_SIZE</td> <td>"ONLY" TCP window size.</td> </tr> <tr> <td>FRAME_URGENT_POINTER</td> <td>"ONLY" TCP urgent pointer.</td> </tr> <tr> <td>FRAME_HARDWARE_TYPE</td> <td>"ALL FOLLOWED:" ARP, RARP hardware type.</td> </tr> <tr> <td>FRAME_HEADER_TYPE</td> <td>"ALL FOLLOWED:" ICMP Header type, IPX Header Type.</td> </tr> <tr> <td>FRAME_HARDWARE_SIZE</td> <td>"ALL FOLLOWED:" ARP, RARP hardware size.</td> </tr> <tr> <td>FRAME_PROTOCOL_TYPE</td> <td>"ALL FOLLOWED:" ARP, RARP protocol type.</td> </tr> <tr> <td>FRAME_PROTOCOL_SIZE</td> <td>"ALL FOLLOWED:" ARP, RARP protocol size.</td> </tr> <tr> <td>FRAME_OPERATION</td> <td>"ALL FOLLOWED:" ARP, RARP operations.</td> </tr> <tr> <td>FRAME_HEADER_CODE</td> <td>"ONLY" ICMP Header codes.</td> </tr> <tr> <td>FRAME_IDENTIFIER</td> <td>"ONLY" ICMP Identifier.</td> </tr> <tr> <td>FRAME_SEN_MAC_ADDR</td> <td>"ANY" protocol MAC sender address.</td> </tr> <tr> <td>FRAME_REC_MAC_ADDR</td> <td>"ANY" protocol MAC receiver address.</td> </tr> <tr> <td>FRAME_HOP</td> <td>"ONLY" IPX Hop</td> </tr> <tr> <td>FRAME_DST_SOCKET</td> <td>"ONLY" IPX destination socket.</td> </tr> <tr> <td>FRAME_SRC_SOCKET</td> <td>"ONLY" IPX source socket.</td> </tr> <tr> <td>FRAME_ICMP_HEADER_CRC</td> <td>"ONLY" ICMP Header checksum.</td> </tr> </table>	FRAME_VERSION	"ONLY" IP version.	FRAME_HEADER_LENGTH	"ALL FOLLOWED:" IP length, IPX length.	FRAME_UDP_HEADER_LENGTH	"ONLY" UDP length.	FRAME_TCP_HEADER_LENGTH	"ONLY" TCP length.	FRAME_TYPE_SERVICE	"ONLY" IP type of service.	FRAME_TOTAL_LENGTH	"ONLY" IP total length.	FRAME_SEQUENCE	"ALL FOLLOWED:" IP sequence, ICMP sequence.	FRAME_UDP_SEQUENCE	"ONLY" UDP sequence.	FRAME_TCP_SEQUENCE	"ONLY" TCP sequence.	FRAME_FLAGS	"ONLY" IP flags.	FRAME_FRAGMENTS_OFFSET	"ONLY" IP fragment and offset.	FRAME_TIME_TO_LIVE	"ONLY" IP time to live.	FRAME_PROTOCOL	"ONLY" IP protocol.	FRAME_HEADER_CRC	"ALL FOLLOWED:" IP checksum, IPX checksum.	FRAME_UDP_HEADER_CRC	"ONLY" UDP checksum.	FRAME_TCP_HEADER_CRC	"ONLY" TCP checksum.	FRAME_DST_IP_ADDR	"ANY" frame Destination IP Address.	FRAME_SRC_IP_ADDR	"ANY" frame Source IP Address.	FRAME_SRC_PORT	"ANY" frame Source Port number	FRAME_DST_PORT	"ANY" frame Destination Port number.	FRAME_ACKNOWLEDGE	"ONLY" TCP Acknowledge number.	FRAME_RESERVED	"ONLY" TCP reserved bits.	FRAME_URG_BIT	"ONLY" TCP URG bit.	FRAME_ACK_BIT	"ONLY" TCP ACK bit.	FRAME_PSH_BIT	"ONLY" TCP PSH bit.	FRAME_RST_BIT	"ONLY" TCP RST bit.	FRAME_SYN_BIT	"ONLY" TCP SYN bit.	FRAME_FIN_BIT	"ONLY" TCP FIN bit.	FRAME_WINDOW_SIZE	"ONLY" TCP window size.	FRAME_URGENT_POINTER	"ONLY" TCP urgent pointer.	FRAME_HARDWARE_TYPE	"ALL FOLLOWED:" ARP, RARP hardware type.	FRAME_HEADER_TYPE	"ALL FOLLOWED:" ICMP Header type, IPX Header Type.	FRAME_HARDWARE_SIZE	"ALL FOLLOWED:" ARP, RARP hardware size.	FRAME_PROTOCOL_TYPE	"ALL FOLLOWED:" ARP, RARP protocol type.	FRAME_PROTOCOL_SIZE	"ALL FOLLOWED:" ARP, RARP protocol size.	FRAME_OPERATION	"ALL FOLLOWED:" ARP, RARP operations.	FRAME_HEADER_CODE	"ONLY" ICMP Header codes.	FRAME_IDENTIFIER	"ONLY" ICMP Identifier.	FRAME_SEN_MAC_ADDR	"ANY" protocol MAC sender address.	FRAME_REC_MAC_ADDR	"ANY" protocol MAC receiver address.	FRAME_HOP	"ONLY" IPX Hop	FRAME_DST_SOCKET	"ONLY" IPX destination socket.	FRAME_SRC_SOCKET	"ONLY" IPX source socket.	FRAME_ICMP_HEADER_CRC	"ONLY" ICMP Header checksum.
FRAME_VERSION	"ONLY" IP version.																																																																																								
FRAME_HEADER_LENGTH	"ALL FOLLOWED:" IP length, IPX length.																																																																																								
FRAME_UDP_HEADER_LENGTH	"ONLY" UDP length.																																																																																								
FRAME_TCP_HEADER_LENGTH	"ONLY" TCP length.																																																																																								
FRAME_TYPE_SERVICE	"ONLY" IP type of service.																																																																																								
FRAME_TOTAL_LENGTH	"ONLY" IP total length.																																																																																								
FRAME_SEQUENCE	"ALL FOLLOWED:" IP sequence, ICMP sequence.																																																																																								
FRAME_UDP_SEQUENCE	"ONLY" UDP sequence.																																																																																								
FRAME_TCP_SEQUENCE	"ONLY" TCP sequence.																																																																																								
FRAME_FLAGS	"ONLY" IP flags.																																																																																								
FRAME_FRAGMENTS_OFFSET	"ONLY" IP fragment and offset.																																																																																								
FRAME_TIME_TO_LIVE	"ONLY" IP time to live.																																																																																								
FRAME_PROTOCOL	"ONLY" IP protocol.																																																																																								
FRAME_HEADER_CRC	"ALL FOLLOWED:" IP checksum, IPX checksum.																																																																																								
FRAME_UDP_HEADER_CRC	"ONLY" UDP checksum.																																																																																								
FRAME_TCP_HEADER_CRC	"ONLY" TCP checksum.																																																																																								
FRAME_DST_IP_ADDR	"ANY" frame Destination IP Address.																																																																																								
FRAME_SRC_IP_ADDR	"ANY" frame Source IP Address.																																																																																								
FRAME_SRC_PORT	"ANY" frame Source Port number																																																																																								
FRAME_DST_PORT	"ANY" frame Destination Port number.																																																																																								
FRAME_ACKNOWLEDGE	"ONLY" TCP Acknowledge number.																																																																																								
FRAME_RESERVED	"ONLY" TCP reserved bits.																																																																																								
FRAME_URG_BIT	"ONLY" TCP URG bit.																																																																																								
FRAME_ACK_BIT	"ONLY" TCP ACK bit.																																																																																								
FRAME_PSH_BIT	"ONLY" TCP PSH bit.																																																																																								
FRAME_RST_BIT	"ONLY" TCP RST bit.																																																																																								
FRAME_SYN_BIT	"ONLY" TCP SYN bit.																																																																																								
FRAME_FIN_BIT	"ONLY" TCP FIN bit.																																																																																								
FRAME_WINDOW_SIZE	"ONLY" TCP window size.																																																																																								
FRAME_URGENT_POINTER	"ONLY" TCP urgent pointer.																																																																																								
FRAME_HARDWARE_TYPE	"ALL FOLLOWED:" ARP, RARP hardware type.																																																																																								
FRAME_HEADER_TYPE	"ALL FOLLOWED:" ICMP Header type, IPX Header Type.																																																																																								
FRAME_HARDWARE_SIZE	"ALL FOLLOWED:" ARP, RARP hardware size.																																																																																								
FRAME_PROTOCOL_TYPE	"ALL FOLLOWED:" ARP, RARP protocol type.																																																																																								
FRAME_PROTOCOL_SIZE	"ALL FOLLOWED:" ARP, RARP protocol size.																																																																																								
FRAME_OPERATION	"ALL FOLLOWED:" ARP, RARP operations.																																																																																								
FRAME_HEADER_CODE	"ONLY" ICMP Header codes.																																																																																								
FRAME_IDENTIFIER	"ONLY" ICMP Identifier.																																																																																								
FRAME_SEN_MAC_ADDR	"ANY" protocol MAC sender address.																																																																																								
FRAME_REC_MAC_ADDR	"ANY" protocol MAC receiver address.																																																																																								
FRAME_HOP	"ONLY" IPX Hop																																																																																								
FRAME_DST_SOCKET	"ONLY" IPX destination socket.																																																																																								
FRAME_SRC_SOCKET	"ONLY" IPX source socket.																																																																																								
FRAME_ICMP_HEADER_CRC	"ONLY" ICMP Header checksum.																																																																																								

	<p><i>PucBytes</i> unsigned char* Pointer to the replacement bytes used to modify the frame component.</p> <p><i>InumBytes</i> int Length of new segment (<i>pucBytes</i>). An error will result if this value does not match the number of bytes being replaced.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Use this function after creating a frame with either <code>NSCreateFrame</code> or <code>NSCreateFrameAndPayload</code> . Related functions: <code>HTFrame</code> and <code>HSDeleteFrame</code> .

NSSetDefaultsFile

Description	Specifies which file to use for default values of Message Function parameters.
Syntax	<code>int NSDefaultsFile(char* szFileName);</code>
Parameters	<p><i>szFileName</i> char* Pointer to a string which can either be the name of the defaults file to use, or the full path of the defaults file.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	See also: <code>HTDefaultStructure</code> and <code>NSEnableAutoDefault</code>

NSSetPayload

Description	Used in conjunction with <code>NSCreateFrame</code> ; this function configures the customized payload (background pattern).
Syntax	<code>Long NSSetPayload(long lFrameID, int iSize, unsigned char* pucPayload)</code>
Parameters	<p><i>lFrameID</i> long The FrameID number is unique for each frame prototype. It is returned by <code>NSCreateFrame</code> and <code>NSCreateFrameAndPayload</code>.</p> <p><i>iSize</i> int The size of the array specifying the payload.</p> <p><i>pucPayload</i> unsigned char* The pointer to the array specifying the payload (the background pattern).</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p><code>NSSetPayload</code> is only used in conjunction with <code>NSCreateFrame</code>, when the value of <code>iPattern</code> (in the structure <code>FrameSpec</code>) is <code>PAT_CUST</code>. This causes <code>NSCreateFrame</code> to not specify a background pattern.</p> <p>Other pre-created payload patterns are available. However, when <code>PAT_CUST</code> is specified, use <code>NSSetPayload</code> to specify a customized pattern.</p> <p>You can also use <code>NSCreateFrameAndPayload</code> to accomplish the same task.</p> <p>Related functions: <code>HTFrame</code>, <code>NSDeleteFrame</code>, and <code>NSModifyFrame</code>.</p>

NSSetPortMappingMode

Description	Sets the port mapping mode for a given link.
Syntax	int NSSetPortMappingMode(int iMode)
Parameters	<p><i>iMode</i> int Specifies which port mapping mode is set.: compatible (like the SMB-2000) or Native for the SMB-6000 platform.</p> <p>PORT_MAPPING_COMPATIBLE Maps like an SMB-2000 PORT_MAPPING_NATIVE Supports full SMB-6000 mapping.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code of less than zero is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	NSSocketLink defaults to Native mode. ETLink and ETSocketLink default to Compatible mode.

NSSocketLink

Description	<p>Establishes a connection to the specified SmartBits chassis using an IP socket connection. Optionally reserves slots using the requested mode.</p> <p><i>Note:</i> Use a serial port connection to configure the IP address of the SMB-6000/600. Refer to the <i>SmartBits 600/6000 Getting Started</i> for steps.</p>
Syntax	int NSSocketLink (char IPAddr, int iTCPPort, int iReserve)
Parameters	<p><i>szIPAddr</i> char Specifies the IP address of the SmartBits chassis to which a connection attempt should be made.</p> <p><i>iTCPPort</i> int TCP port number. The default value is 16385.</p> <p><i>iReserve</i> int Slots to reserve initially. Possible values are:</p> <p>RESERVE_NONE Reserve none of the slots in the chassis.</p> <p>RESERVE_ALL Reserve all the slots in the chassis.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code (less than zero) is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>This command may be used only with SmartBits 6000/600 chassis.</p> <p>This function connects to the SmartBits chassis using the provided IP address and port ID. The IP address must be defined as a dotted-decimal string. Host names are not resolved.</p> <p>The <i>iReserve</i> parameter controls the initial reservation of slots as follows: RESERVE_NONE causes no slots to be reserved after the link is established. RESERVE_ALL causes all slots to be reserved after the link is established.</p> <p>Once connected, the port-mapping mode is set to PORT_MAPPING_NATIVE.</p> <p>Linking to the same chassis more than once will succeed for a system that is multi-user capable (for Release 3.06b, only the SmartBits 6000/600). Attempting to link twice to a system that is not multi-user capable, or to the serial port on a multi-user-capable system, will fail.</p> <p>After linking, the hub number of the first chassis on the link will be equal to the hub number of the previous link's first chassis plus 4.</p>

NSUnLink

Description	Severs a connection to the current link.
Syntax	int NSUnLink ()
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code (less than zero) is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>This function closes the current link, which may be either the most recently established link or the last link specified by an ETSetCurrentLink command or an ETSetCurrentSockLink command.</p> <p>After unlinking, all chassis with hub numbers higher than hubs in the severed link will have their hub numbers reassigned. Each new hub number will be decreased by 4 from its previous value.</p>

Appendix A:

Original Functions for the ET-1000 Only

Some Original Functions apply only to the ET-1000 chassis, a precursor to the current SmartBits chassis. They are supported by the library but are not generally used. This appendix lists and describes these ET-1000-only functions (all begin with the prefix **ET**).

The table below is a summary. Detailed descriptions follow the table.

Original Functions for the ET-1000 Only	
ETAlignCount	ETGetGapScale
ETBNC	ETGetJET210Mode
ETBurst	ETGetLNM
ETCaptureParams	ETGetPreamble
ETCaptureRun	ETGetReceiveTrigger
ETCollision	ETGetRun
ETDataLength	ETGetSel
ETDataPatternx	ETGetSwitch
ETDribbleCountx	ETGetTransmitTrigger
ETGap	ETGetVFDRun
ETGapScale	ETLNM
ETGetAlignCount	ETLoopback
ETGetBNC	ETMFCounter
ETGetBurstCount	ETPreamblex
ETGetBurstMode	ETReceiveTrigger
ETGetCapturePacket	ETRemotex
ETGetCapturePacketCount	ETReset
ETGetCaptureParams	ETReturnAddress
ETGetCollision	ETRun
ETGetCounters	ETSetJET210Mode
ETGetCRCErrors	ETSetSel
ETGetCurrentLink	ETSetup
ETGetDataLength	ETTransmitCRC
ETGetDataPattern	ETTransmitTrigger
ETGetDribbleCount	ETVFDParams
ETGetErrorStatus	ETVFDRun
ETGetGap	

ETAlignCount

Description	Specifies the number of alignment error bits to insert into the transmit stream. This is used to generate alignment errors. If Count is zero, then alignment errors are not introduced into the transmit stream.
Syntax	int ETAlignCount(int Count)
Parameters	<i>Count</i> int Specifies the number of alignment error bits to introduce into every transmitted packet. <i>Range:</i> 0 to 7. Numbers outside this range are invalid and have no effect on the alignment error count.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

ETBNC

Description	Defines the function associated with the rear panel BNC connectors.																																																								
Syntax	int ETBNC(int BNCid, int Config)																																																								
Parameters	<p>BNCid int Identifies the rear panel BNC connector being addressed.</p> <p style="padding-left: 40px;">ETBNC_1 = BNC#1 ETBNC_2 = BNC#2 ETBNC_3 = BNC#3</p> <p>All other values are invalid and have no effect on the current BNC mode.</p> <p>Config int Identifies the specific function associated with the BNC. The following arguments are valid:</p> <table style="width: 100%; border: none;"> <tr><td style="padding-left: 40px;">ETBNC_INPUT</td><td>Input (Hi-Z)</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RXEA</td><td>Receive enable, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RXEB</td><td>Receive enable, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RCKA</td><td>Receive Clock, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RCKB</td><td>Receive Clock, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RDATA</td><td>Receive Data, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RDATB</td><td>Receive Data, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_TXEA</td><td>Transmit Enable, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_TXEB</td><td>Transmit Enable, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_TDAT</td><td>Transmit Data</td></tr> <tr><td style="padding-left: 40px;">ETBNC_COLLISIONA</td><td>Collision, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_COLLISIONB</td><td>Collision, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_CRCA</td><td>CRC Error, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_CRCB</td><td>CRC Error, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_UNDRA</td><td>Undersize Error, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_UNDRB</td><td>Undersize Error, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_OVRA</td><td>Oversize Error, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_OVRB</td><td>Oversize Error, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_ALA</td><td>Alignment Error, Port A</td></tr> <tr><td style="padding-left: 40px;">ETBNC_ALB</td><td>Alignment Error, Port B</td></tr> <tr><td style="padding-left: 40px;">ETBNC_TXTRIG</td><td>Transmit Trigger</td></tr> <tr><td style="padding-left: 40px;">ETBNC_RXTRIG</td><td>Receive Trigger</td></tr> <tr><td style="padding-left: 40px;">ETBNC_10MHZ</td><td>10 MHz internal clock</td></tr> <tr><td style="padding-left: 40px;">ETBNC_10MHZINV</td><td>10 MHz internal clock, inverted</td></tr> <tr><td style="padding-left: 40px;">ETBNC_20MHZ</td><td>20 MHz internal clock</td></tr> <tr><td style="padding-left: 40px;">ETBNC_20MHZINV</td><td>20 MHz internal clock, inverted</td></tr> <tr><td style="padding-left: 40px;">ETBNC_EXTCLK</td><td>External Clock input, BNC#3 only</td></tr> <tr><td style="padding-left: 40px;">ETBNC_EXTCLKINV</td><td>External Clock inverted input, BNC#3 only</td></tr> </table> <p>All other values are invalid and have no effect on the current BNC mode</p>	ETBNC_INPUT	Input (Hi-Z)	ETBNC_RXEA	Receive enable, Port A	ETBNC_RXEB	Receive enable, Port B	ETBNC_RCKA	Receive Clock, Port A	ETBNC_RCKB	Receive Clock, Port B	ETBNC_RDATA	Receive Data, Port A	ETBNC_RDATB	Receive Data, Port B	ETBNC_TXEA	Transmit Enable, Port A	ETBNC_TXEB	Transmit Enable, Port B	ETBNC_TDAT	Transmit Data	ETBNC_COLLISIONA	Collision, Port A	ETBNC_COLLISIONB	Collision, Port B	ETBNC_CRCA	CRC Error, Port A	ETBNC_CRCB	CRC Error, Port B	ETBNC_UNDRA	Undersize Error, Port A	ETBNC_UNDRB	Undersize Error, Port B	ETBNC_OVRA	Oversize Error, Port A	ETBNC_OVRB	Oversize Error, Port B	ETBNC_ALA	Alignment Error, Port A	ETBNC_ALB	Alignment Error, Port B	ETBNC_TXTRIG	Transmit Trigger	ETBNC_RXTRIG	Receive Trigger	ETBNC_10MHZ	10 MHz internal clock	ETBNC_10MHZINV	10 MHz internal clock, inverted	ETBNC_20MHZ	20 MHz internal clock	ETBNC_20MHZINV	20 MHz internal clock, inverted	ETBNC_EXTCLK	External Clock input, BNC#3 only	ETBNC_EXTCLKINV	External Clock inverted input, BNC#3 only
ETBNC_INPUT	Input (Hi-Z)																																																								
ETBNC_RXEA	Receive enable, Port A																																																								
ETBNC_RXEB	Receive enable, Port B																																																								
ETBNC_RCKA	Receive Clock, Port A																																																								
ETBNC_RCKB	Receive Clock, Port B																																																								
ETBNC_RDATA	Receive Data, Port A																																																								
ETBNC_RDATB	Receive Data, Port B																																																								
ETBNC_TXEA	Transmit Enable, Port A																																																								
ETBNC_TXEB	Transmit Enable, Port B																																																								
ETBNC_TDAT	Transmit Data																																																								
ETBNC_COLLISIONA	Collision, Port A																																																								
ETBNC_COLLISIONB	Collision, Port B																																																								
ETBNC_CRCA	CRC Error, Port A																																																								
ETBNC_CRCB	CRC Error, Port B																																																								
ETBNC_UNDRA	Undersize Error, Port A																																																								
ETBNC_UNDRB	Undersize Error, Port B																																																								
ETBNC_OVRA	Oversize Error, Port A																																																								
ETBNC_OVRB	Oversize Error, Port B																																																								
ETBNC_ALA	Alignment Error, Port A																																																								
ETBNC_ALB	Alignment Error, Port B																																																								
ETBNC_TXTRIG	Transmit Trigger																																																								
ETBNC_RXTRIG	Receive Trigger																																																								
ETBNC_10MHZ	10 MHz internal clock																																																								
ETBNC_10MHZINV	10 MHz internal clock, inverted																																																								
ETBNC_20MHZ	20 MHz internal clock																																																								
ETBNC_20MHZINV	20 MHz internal clock, inverted																																																								
ETBNC_EXTCLK	External Clock input, BNC#3 only																																																								
ETBNC_EXTCLKINV	External Clock inverted input, BNC#3 only																																																								
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>																																																								
Comments	<p>If the JET-210 mode had previously been active, then the execution of this function for BNCid will place BNCid in the requested mode and the other two BNCid's in the input mode. Conversely, any subsequent execution of the <i>SetJET210Mode(1)</i> function will place all three BNCid's in the JET-210 mode.</p> <p>ADVICE: When in doubt, use function <i>ETGetBNC(...)</i> to find out specifically what mode the BNC's are in.</p>																																																								

ETBurst

Description	Specifies the Burst Mode and the Burst Count
Syntax	int ETBurst(int Mode, long Count)
Parameters	<p><i>Mode</i> int Identifies whether or not the Burst Mode is on or off: ETBURST_ONBurst mode ON ETBURST_OFF Burst mode OFF All other values are invalid and have no effect on the current burst mode.</p> <p><i>Count</i> long Specifies the number of packets to be transmitted during the Burst. Range: 1 to $2^{24}-1$ (1-16777215) All values outside this range are invalid and have no effect on the current burst mode.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	Once the Burst Mode is enabled, the ETRun function takes on a different characteristic: "Step" causes the ET-1000 to internally load the Burst Count. "Run" causes the ET-1000 to either transmit the number of packets previously loaded (using "Step") <i>OR</i> transmit a single packet if no internal Burst Counts were previously loaded.

ETCaptureParams

Description	Specifies Capture Offset, Range, Filter, Port, Buffer mode, Time-tag and run mode. All parameters must be put into CStruct before calling this function.
Syntax	int ETCaptureParams(CaptureStructure* CStruct)
Parameters	<i>CStruct</i> CaptureStructure* Points to the CStruct structure that holds all the capture parameters. The structure must be loaded before calling this routine. If CStruct contains values outside appropriate ranges, this function will not execute.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	See CaptureStructure definition below.

CaptureStructure

Parameter	Description
unsigned Offset	Integer value specifying the offset (in bit times) from the first bit after the preamble. This value is returned as 0 in ETGetCaptureParams if Mode is CAPTURE_ENTIRE_PACKET. <i>Range: 0 to 65535 (0x0000 to 0xFFFF)</i>
unsigned Range	Integer value specifying the number of bits to capture within each packet, once the capture criteria have been met. <i>Range: 0 to 65535 (0x0000 to 0xFFFF).</i> If Range is larger than the packet size, capturing on that packet is halted at the end of the packet. This value is returned as 0 in ETGetCaptureParams if Mode is CAPTURE_ENTIRE_PACKET.
int Filter	Specifies the type of data to capture and filter. The Filter type can be one of the following or a combination. To get a combination, create an integer by “OR-ing” together criteria from the list. Remember that the Range and Offset values still apply. Thus when All Data is selected, only the data that satisfies the Range and Offset criteria is captured and stored. <ul style="list-style-type: none"> CAPTURE_NONE None (off) CAPTURE_ANY Any data on the line CAPTURE_NOT_GOOD Non standard Ethernet packets CAPTURE_GOOD Packets without error CAPTURE_ERRS_RXTRIG Packets with any following errors (same as previous version’s “All Data”) CAPTURE_RXTRIG Specified by Receive Trigger CAPTURE_CRC CRC errored packets CAPTURE_ALIGN Alignment errored packets CAPTURE_OVERSIZE Oversize packets CAPTURE_UNDERSIZE Undersize packets CAPTURE_COLLISION Collision packets
int Port	Identifies the port used in capturing data. <ul style="list-style-type: none"> PORT_A Port A PORT_B Port B
int BufferMode	Specifies how the capture buffer is to be used. Continuous capture. When the capture buffer fills up, it continues capturing data, which overwrites the previously captured data. One-shot. When the capture buffer fills up, capture is stopped.
int TimeTag	This value must always be OFF to get valid capture data. (Use of TIME_TAG_ON will result in unpredictable results): Time tagging is disabled.
int Mode	Determines the capture mode: <ul style="list-style-type: none"> CAPTURE_ENTIRE_PACKET Capture all data. CAPTURE_RANGE Capture only the portions of packets specified by Range and Offset. CAPTURE_OFF Off (no capture)

ETCaptureRun

Description	Starts (or restarts) the capture process.
Syntax	int ETCaptureRun(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See <i>Appendix B for error codes</i> .
Comments	It is advised that you set up the desired capture parameters with the ETCaptureParams(CaptureStructure *CStruct) function before calling this function. Otherwise, the attached ET-1000 will run whatever capture sequence was previously left in it. Use the ETGetCapturePacketCount function to monitor the number of packets successfully captured after you initiate the capture process with this command. Use the ETGetCapturePacket(...) function to retrieve packets captured. To clear the buffer, you must turn the Capture off and then back on. If a capture is currently in progress when this function is executed, all captured data obtained thus far will be discarded and replaced with new capture information. See the definition of CaptureStructure above.

ETCollision

Description	Determines the collision mode, offset, duration, and count.
Syntax	int ETCollision(CollisionStructure* CStruct)
Parameters	<i>CStruct</i> CollisionStructure* Holds information pertaining to the collision mode (off, long, adjustable, Port A receive packet or Port B receive packet), the collision offset (in bits), duration (bit-times) and count.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See <i>Appendix B for error codes</i> .
Comments	See the definition of CollisionStructure below.

CollisionStructure

Parameter	Description										
unsigned Offset	<p>Specifies the offset, in bits, starting from the first bit of the preamble where the collision is to take place. This value is only used when the Collision Mode is COLLISION_ADJ, CORP_A or CORP_B. It is ignored when the Collision Mode is COLLISION_LONG.</p> <p><i>Range:</i> 0 to 65535 (0x0000 to 0xFFFF).</p> <p>Note that the Offset value entered here also pertains to the collisions produced on the SmartBits when it is attached to the ET-1000.</p>										
unsigned Duration	<p>Specifies the duration in bits that the collision is to be asserted. This value is used only when the Collision Mode is COLLISION_ADJ, CORP_A or CORP_B. It is ignored when the Collision Mode is COLLISION_LONG.</p> <p><i>Range:</i> 1 to 65535 (0x0000 to 0xFFFF). A duration of 0 is invalid.</p> <p>If Note that the Duration value entered here also pertains to the collisions produced on the SmartBits when it is attached to the ET-1000.</p>										
int Count	<p>Specifies the number of consecutive collisions to produce (one in each packet) before the collision goes inactive. A count of 0 essentially disables the collision counting mechanism, thus producing continuous collisions of the specified type.</p> <p><i>Range:</i> 0 to 1024.</p> <p>If Note that the Duration value entered here also pertains to the collisions produced on the SmartBits when it is attached to the ET-1000.</p>										
int Mode	<p>Specifies the collision mode.</p> <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;">COLLISION_OFF</td> <td>Collision Off</td> </tr> <tr> <td style="padding-left: 20px;">COLLISION_LONG</td> <td>Long Collision</td> </tr> <tr> <td style="padding-left: 20px;">COLLISION_ADJ</td> <td>Adjustable Collision (on transmission)</td> </tr> <tr> <td style="padding-left: 20px;">CORP_A</td> <td>Collision on receive packet, Port A</td> </tr> <tr> <td style="padding-left: 20px;">CORP_B</td> <td>Collision on receive packet, Port B</td> </tr> </table>	COLLISION_OFF	Collision Off	COLLISION_LONG	Long Collision	COLLISION_ADJ	Adjustable Collision (on transmission)	CORP_A	Collision on receive packet, Port A	CORP_B	Collision on receive packet, Port B
COLLISION_OFF	Collision Off										
COLLISION_LONG	Long Collision										
COLLISION_ADJ	Adjustable Collision (on transmission)										
CORP_A	Collision on receive packet, Port A										
CORP_B	Collision on receive packet, Port B										

ETDataLength

Description	Specifies the number of bytes per packet to be used in transmitting data from the ET-1000.
Syntax	int ETDataLength(long Count)
Parameters	<p><i>Count</i> long Contains the number of bytes that are to be inserted in each packet.</p> <p>Range: 0 to 999,999. Values outside this range are invalid and have no effect on the transmitted data length.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. See Appendix B for error codes.
Comments	<i>Count</i> does not include the 4 CRC bytes appended to every normal Ethernet packet.

ETDataPattern

Description	Defines the background data pattern to transmit.																																		
Syntax	int ETDataPattern(int Pattern)																																		
Parameters	<p><i>Pattern</i> int Determines the type of pattern that is transmitted out Port A and/or Port B. The choices are:</p> <table border="0"> <tr><td>ETDP_ALLZERO</td><td>All 0</td></tr> <tr><td>ETDP_ALLONE</td><td>All 1</td></tr> <tr><td>ETDP_RANDOM</td><td>Random</td></tr> <tr><td>ETDP_AAAA</td><td>Continuous AAAA (hex)</td></tr> <tr><td>ETDP_5555</td><td>Continuous 5555 (hex)</td></tr> <tr><td>ETDP_F0F0</td><td>Continuous F0F0 (hex)</td></tr> <tr><td>ETDP_0F0F</td><td>Continuous 0F0F (hex)</td></tr> <tr><td>ETDP_00FF</td><td>Continuous 00FF00FF (hex)</td></tr> <tr><td>ETDP_FF00</td><td>Continuous FF00FF00 (hex)</td></tr> <tr><td>ETDP_0000FFFF</td><td>Continuous 0000FFFF0000FFFF (hex)</td></tr> <tr><td>ETDP_FFFF0000</td><td>Continuous FFFF0000FFFF0000 (hex)</td></tr> <tr><td>ETDP_00000000FFFFFFFF</td><td>Continuous 00000000FFFFFFFF (hex)</td></tr> <tr><td>ETDP_FFFFFFFF00000000</td><td>Continuous FFFFFFFF00000000 (hex)</td></tr> <tr><td>ETDP_INCR8</td><td>Incrementing 8 bit pattern</td></tr> <tr><td>ETDP_INCR16</td><td>Incrementing 16 bit pattern</td></tr> <tr><td>ETDP_DECR8</td><td>Decrementing 8 bit pattern</td></tr> <tr><td>ETDP_DECR16</td><td>Decrementing 16 bit pattern</td></tr> </table> <p>All other values are invalid and will result in no changes to the currently transmitted data pattern</p>	ETDP_ALLZERO	All 0	ETDP_ALLONE	All 1	ETDP_RANDOM	Random	ETDP_AAAA	Continuous AAAA (hex)	ETDP_5555	Continuous 5555 (hex)	ETDP_F0F0	Continuous F0F0 (hex)	ETDP_0F0F	Continuous 0F0F (hex)	ETDP_00FF	Continuous 00FF00FF (hex)	ETDP_FF00	Continuous FF00FF00 (hex)	ETDP_0000FFFF	Continuous 0000FFFF0000FFFF (hex)	ETDP_FFFF0000	Continuous FFFF0000FFFF0000 (hex)	ETDP_00000000FFFFFFFF	Continuous 00000000FFFFFFFF (hex)	ETDP_FFFFFFFF00000000	Continuous FFFFFFFF00000000 (hex)	ETDP_INCR8	Incrementing 8 bit pattern	ETDP_INCR16	Incrementing 16 bit pattern	ETDP_DECR8	Decrementing 8 bit pattern	ETDP_DECR16	Decrementing 16 bit pattern
ETDP_ALLZERO	All 0																																		
ETDP_ALLONE	All 1																																		
ETDP_RANDOM	Random																																		
ETDP_AAAA	Continuous AAAA (hex)																																		
ETDP_5555	Continuous 5555 (hex)																																		
ETDP_F0F0	Continuous F0F0 (hex)																																		
ETDP_0F0F	Continuous 0F0F (hex)																																		
ETDP_00FF	Continuous 00FF00FF (hex)																																		
ETDP_FF00	Continuous FF00FF00 (hex)																																		
ETDP_0000FFFF	Continuous 0000FFFF0000FFFF (hex)																																		
ETDP_FFFF0000	Continuous FFFF0000FFFF0000 (hex)																																		
ETDP_00000000FFFFFFFF	Continuous 00000000FFFFFFFF (hex)																																		
ETDP_FFFFFFFF00000000	Continuous FFFFFFFF00000000 (hex)																																		
ETDP_INCR8	Incrementing 8 bit pattern																																		
ETDP_INCR16	Incrementing 16 bit pattern																																		
ETDP_DECR8	Decrementing 8 bit pattern																																		
ETDP_DECR16	Decrementing 16 bit pattern																																		
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>																																		
Comments	If VFD is active, then its pattern will be transmitted for the duration and offset specified in the applicable VFDStructure. Any transmitted data outside this envelope will consist of the data pattern specified in this function.																																		

ETDribbleCount

Description	Specifies the number of dribble bits to insert into the transmit stream.
Syntax	int ETDribbleCount(int Count)
Parameters	<p><i>Count</i> int Determines the number of dribble bits to insert.</p> <p>Range: 0 to 7. A value of 0 inserts no dribble bits. Any value outside this range is invalid and will result in no changes to the current dribble count.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

ETGap

Description	Specifies the inter-packet gap value that is to be transmitted.
Syntax	int ETGap(long Count)
Parameters	<p><i>Count</i> long Determines the gap value to be inserted in the transmit stream of both ports.</p> <p>Range: 0 to 999,999. Any values outside this range are invalid and result in no changes to the current gap setting.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>The value of <i>Count</i> is further scaled by the most recent value left in function ETGapScale(int TimeOfGap). If the scale is set to the "100ns" setting, then the number left in <i>Count</i> will produce an inter-packet gap according to the following formula:</p> $\text{GAP} = 600 + (100 * \text{Count}) \text{ nanoseconds}$ <p>If the scale is set to the "1μs" setting, then the number left in <i>Count</i> will produce an inter-packet gap according to the following formula:</p> $\text{GAP} = 0.6 + \text{Count} \text{ microseconds}$ <p>The ETGap and ETGapScale functions may appear in any order; however, keep in mind that the attached ET-1000 will execute each instruction in the order in which it is received. Thus, setting the scale before setting the Gap value will result in the sending of two or more consecutive packets with an interim value for the gap. To avoid this problem, stop transmission (ETRun function) before changing the Gap parameters, and then re-start transmission when done.</p>

ETGapScale

Description	Specifies that either a 100 ns gap scale or a 1 μ s gap scale is to be used in determining the gap time.				
Syntax	int ETGapScale(int TimeOfGap)				
Parameters	<p><i>TimeOfGap</i> int Determines the scale to be used for setting the gap time:</p> <table style="margin-left: 20px;"> <tr> <td>ETGAP_100NS</td> <td>100 nanosecond gap scale</td> </tr> <tr> <td>ETGAP_1US</td> <td>1 microsecond gap scale</td> </tr> </table> <p>All other values are invalid and will result in no changes to the gap scale setting.</p>	ETGAP_100NS	100 nanosecond gap scale	ETGAP_1US	1 microsecond gap scale
ETGAP_100NS	100 nanosecond gap scale				
ETGAP_1US	1 microsecond gap scale				
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>				
Comments	See the comment section under function ETGap(long Count) .				

ETGetAlignCount

Description	Returns the number of alignment error bits currently being inserted into the transmit data stream.
Syntax	int ETGetAlignCount(void)
Parameters	None
Return Value	The return value will range from 0 to 7, corresponding to the number of alignment error bits being inserted. If the return value is less than zero, a failure occurred. <i>See Appendix B for error codes.</i>
Comments	To set the number of alignment error bits for transmission, use function ETAlignCount .

ETGetBNC

Description	Retrieves the configuration of the BNC identified by BNCid.						
Syntax	int ETGetBNC(int BNCid)						
Parameters	<p><i>BNCid</i> int Identifies the BNC connector whose configuration is needed:</p> <table style="margin-left: 40px;"> <tr> <td>ETBNC_1</td> <td>BNC #1</td> </tr> <tr> <td>ETBNC_2</td> <td>BNC #2</td> </tr> <tr> <td>ETBNC_3</td> <td>BNC #3</td> </tr> </table> <p style="margin-left: 40px;">Any values outside this range are invalid and will return a failure code.</p>	ETBNC_1	BNC #1	ETBNC_2	BNC #2	ETBNC_3	BNC #3
ETBNC_1	BNC #1						
ETBNC_2	BNC #2						
ETBNC_3	BNC #3						
Return Value	The return value corresponds to the most recent command which set the function for the BNC. See ETBNC for an identification of these values. (Note that a return value of 99 indicates that the BNCs are in the JET-210 mode.) If the return value is less than zero, a failure occurred. <i>See Appendix B for error codes.</i>						
Comments	See function ETBNC to set the configuration for a particular BNC.						

ETGetBurstCount

Description	Returns the current Burst Count.
Syntax	long ETGetBurstCount(void)
Parameters	None.
Return Value	Returns the current Burst Count, which ranges from 1 to $2^{24}-1$. If the return value is less than zero, a failure occurred. <i>See Appendix B for error codes.</i>
Comments	The Burst Mode need not be enabled in order to execute this function. See the ETBurst function to establish the burst mode and count.

ETGetBurstMode

Description	Returns the current Burst Mode.
Syntax	int ETGetBurstMode(void)
Parameters	None
Return Value	Returns the current Burst Mode, ranging from ET_OFF (0) to ET_ON (1). If the return value is less than zero, a failure occurred. <i>See Appendix B for error codes.</i>
Comments	See the ETBurst function to establish the burst mode and count.

ETGetCapturePacket

Description	Dumps the data from a captured packet into a specified location.
Syntax	int ETGetCapturePacket(long PI, int far * Buffer, int BufferSize)
Parameters	<p><i>PI</i> long Identifies the packet whose contents are to be read into <i>Buffer</i>. Packet numbers start at zero.</p> <p><i>Buffer</i> int* (far pointer) Points to an area in memory where the packet data is to be placed.</p> <p><i>BufferSize</i> int Determines the maximum number of characters to be put into <i>Buffer</i>.</p>
Return Value	The return value specifies the number of characters written into <i>Buffer</i> (not counting NULL, if any) if the function executed successfully. It will be a positive number greater than or equal to zero. If the return value is less than zero, a failure occurred. <i>See Appendix B for error codes.</i>
Comments	To determine the number of packets before actually retrieving them, use <i>ETGetCapturePacketCount(...)</i> .

ETGetCapturePacketCount

Description	Returns the number of complete packets captured thus far.
Syntax	long ETGetCapturePacketCount(void)
Parameters	None
Return Value	This function returns a long integer if it executed correctly. The integer indicates the number of packets successfully captured by the attached ET-1000. If the return value is less than zero, then it is a failure code. <i>See Appendix B for error codes.</i>
Comments	If in Continuous Capture mode, you must stop capture before getting the <i>CapturePacketCount</i> .

ETGetCaptureParams

Description	Returns the current capture parameters.
Syntax	int ETGetCaptureParams(CaptureStructure* CStruct)
Parameters	<i>CStruct</i> CaptureStructure* Pointer to the CaptureStructure structure that is to hold the capture parameters.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	Use function ETCaptureParams to define the capture parameters on the attached ET-1000. You need not define the capture parameters before calling this function. The information returned in the CaptureStructure structure represents the current setup on the attached ET-1000. See the definition of CaptureStructure on page 201.

ETGetCollision

Description	Returns the current collision mode.
Syntax	int ETGetCollision(CollisionStructure* CStruct)
Parameters	<i>CStruct</i> CollisionStructure* Points to the structure to be filled with information pertaining to the collision setup inside the attached ET-1000.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	See the definition of CollisionStructure in the Data Structures portion of this manual.

ETGetCounters

Description	Retrieves all counter information from the attached ET 1000.
Syntax	int ETGetCounters(CountStructure* CStruct)
Parameters	<i>CStruct</i> CountStructure* Points to the CountStructure structure which is to hold all the information pertaining to the ET-1000's internal counters.
Return Value	The return value is ≥ 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	See the definition of CountStructure below

CountStructure

Parameter	Description
unsigned long ERAEvent	Event count for CRC errors on Port A.
unsigned long ERARate	Rate count for CRC errors on Port A.
unsigned long ERBEvent	Event count for CRC errors on Port B.
unsigned long ERBRate	Rate count for CRC errors on Port B.
unsigned long TXAEvent	Event count for transmitted bits on Port A.
unsigned long TXARate	Rate count for transmitted bits on Port A.
unsigned long TXBEvent	Event count for transmitted bits on Port B.
unsigned long TXBRate	Rate count for transmitted bits on Port B.
unsigned long RXAEvent	Event count for received bits on Port A.
unsigned long RXARate	Rate count for received bits on Port A.
unsigned long RXBEvent	Event count for received bits on Port B.
unsigned long RXBRate	Rate count for received bits on Port B.
unsigned long CXAEvent	Event count for collisions on Port A.
unsigned long CXARate	Rate count for collisions on Port A.
unsigned long CXBEvent	Event count for collision on Port B.
unsigned long CXBRate	Rate count for collisions on Port B.
unsigned long ALAEvent	Event count for alignment errors Port A.
unsigned long ALARate	Rate count for alignment errors Port A.
unsigned long ALBEvent	Event count for alignment errors Port B.
unsigned long ALBRate	Rate count for alignment errors Port B.
unsigned long UPAEvent	Event count for undersize packets on Port A.
unsigned long UPARate	Rate count for undersize packets on Port A.
unsigned long UPBEvent	Event count for undersize packets on Port B.
unsigned long UPBRate	Rate count for undersize packets on Port B.
unsigned long OPAEvent	Event count for oversize packets on Port A.
unsigned long OPARate	Rate count for oversize packets on Port A.
unsigned long OPBEvent	Event count for oversize packets on Port B.
unsigned long OPBEvent	Event count for oversize packets on Port B.
unsigned long OPBRate	Rate count for oversize packets on Port B.
unsigned long MFAEvent	Event Multi-Function count, Port A.
unsigned long MFARate	Rate Multi-Function count, Port A.
unsigned long MFBEvent	Event Multi-Function count, Port B.
unsigned long MFBRate	Rate Multi-Function count, Port B.

ETGetCRCError

Description	Used to inquire whether or not CRC errors are currently being transmitted by the attached ET-1000.
Syntax	int ETGetCRCError(void)
Parameters	None
Return Value	This function returns ET_OFF (0) if CRC errors are currently NOT being transmitted. A value of ET_ON (1) is returned if CRC errors ARE currently being transmitted. A return value less than zero is a failure code. <i>See Appendix B for error codes.</i>
Comments	None

ETGetCurrentLink

Description	Used to inquire which attached ET-1000 in the Programming Library is the current one.
Syntax	int ETGetCurrentLink(void)
Parameters	None
Return Value	This function returns the ET-1000 ComPort which is associated with “current” ET-1000.
Comments	See ETSetCurrentLink, ETLink.

ETGetDataLength

Description	Returns the current length, in bytes, of the transmitted data packet.
Syntax	long ETGetDataLength(void)
Parameters	None
Return Value	This function returns the length, in bytes, of the attached ET-1000’s transmitted data packets. The number does not include the four bytes of CRC. If the function is successful, the returned value will range from 0 to 999,999. A return value less than zero is a failure code, indicating that the function failed. <i>See Appendix B for error codes.</i>
Comments	None

ETGetDataPattern

Description	Returns the identity of the current background transmit data pattern.
Syntax	int ETGetDataPattern(void)
Parameters	None
Return Value	If the function executed successfully, it returns a value corresponding to the current background data pattern. These values have the same meaning as parameter <i>Pattern</i> in function ETDataPattern . This function returns a failure code (less than zero) if it failed. <i>See Appendix B for error codes.</i>
Comments	None

ETGetDribbleCount

Description	Returns the current number of dribble bits being inserted into the transmit stream of the attached ET-1000.
Syntax	int ETGetDribbleCount(void)
Parameters	None
Return Value	Returns the number of dribble bits being inserted. Range: 0 to 7. This function returns a failure code (less than zero) if it failed. <i>See Appendix B for error codes.</i>
Comments	None

ETGetErrorStatus

Description	Used to inquire the nature of the most recent failure on the communications port.
Syntax	int ETGetErrorStatus(void)
Parameters	None
Return Value	The return value indicates the failure code of the most recent serial port failure. <i>See Appendix B for error codes.</i> If no failures have been detected, this function returns a zero.
Comments	See Appendix B to interpret the return value from this function.

ETGetGap

Description	Returns the gap value currently being transmitted by the attached ET-1000.
Syntax	long ETGetGap(void)
Parameters	None
Return Value	Returns the gap value currently in use by the attached ET-1000. Range: 0 to 999,999. This function returns a failure code (less than zero) if it failed. <i>See Appendix B for error codes.</i>
Comments	<p>The correspondence between the gap value and the actual gap time in the ET-1000's transmit stream depends on the current gap scale in use. Use function ETGetGapScale to find out what scale is currently in use.</p> <ul style="list-style-type: none"> • If the scale is set to the "100ns" setting (ETGAP_100NS), then the physical gap value is expressed as: $GAP = 600 + (100 * Return Value)$ nanoseconds • If the scale is set to the "1µs" setting (ETGAP_1US), then the physical gap value is expressed as: $GAP = 9.6 + Return Value$ microseconds.

ETGetGapScale

Description	Returns the current gap scale in use by the attached ET-1000.
Syntax	int ETGetGapScale(void)
Parameters	None
Return Value	If the function is successful, then the return value is 0 when the ET-1000 gap scale is set to the 1 microsecond scale. The return value is 1 when the gap scale is 100 nanoseconds. This function returns a failure code if it failed. <i>See Appendix B for error codes.</i>
Comments	See the comment section of function ETGetGap .

ETGetJET210Mode

Description	Returns the current ET-1000 JET210 mode.
Syntax	int ETGetJET210Mode(void)
Parameters	None
Return Value	ET_OFF JET-210 mode disabled ET_ON JET-210 mode enabled Returns a failure code if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

ETGetLNM

Description	Returns the current Live Network Mode status of the attached ET-1000.
Syntax	int ETGetLNM(void)
Parameters	None
Return Value	The return value is either ETLNM_ON to indicate that the attached ET-1000's Live Network Mode is active, or ETLNM_OFF to indicate that the attached ET-1000's Live Network Mode is inactive. If the return value is neither of these, then an error condition has been detected. The return value will be less than zero in this case, indicating the failure code. <i>See Appendix B for error codes.</i>
Comments	Live Network Mode is currently available only for the ET-1000's Port A.

ETGetPreamble

Description	Returns the current number of preamble bits being inserted into the transmit stream by the attached ET-1000.
Syntax	int ETGetPreamble(void)
Parameters	None
Return Value	Returns the number of preamble bits being used, in the range 10 to 128. A return value less than 0 indicates a failure. <i>See Appendix B for error codes.</i>
Comments	None

ETGetReceiveTrigger

Description	Returns the receive trigger parameters currently implemented by the attached ET-1000.
Syntax	int ETGetReceiveTrigger(TriggerStructure* RStruct)
Parameters	<i>RStruct</i> TriggerStructure* Points to a TriggerStructure structure which is to contain the trigger parameters
Return Value	The return value is >= 0 if the function executed successfully. The return value is < 0 if there was a failure. <i>See Appendix B for error codes.</i>
Comments	See the definition of TriggerStructure below.

TriggerStructure

Parameter	Description
unsigned Offset	Specifies the number of bit times that pass between the first non-preamble bit and when the trigger word is searched for in the data stream. <i>Range: 0 to 65535 (0x0000 to 0xFFFF).</i>
unsigned Range	Specifies the size of the trigger word, in bits. <i>Range: 1 to 96 (0x0001 to 0x0060).</i>
unsigned Pattern	Array of bytes containing the trigger word. Pattern[0] is the LSByte, Pattern[11] is the MSByte. The lower 8 bits of each element contain trigger information. The upper 8 bits are "don't cares."

ETGetRun

Description	Returns the current run state of the attached ET-1000.
Syntax	int ETGetRun(void)
Parameters	None
Return Value	The return value depends on the run state: ETSTOP "Stop" mode ETSTEP "Step" mode ETRUN "Run" mode A return value less than 0 indicates a failure. <i>See Appendix B for error codes.</i>
Comments	

ETGetSel

Description	Returns the current Select state of the attached ET-1000.
Syntax	int ETGetSel(void)
Parameters	None
Return Value	Return value depends on the current Select state: ETSELA Transmit on A, receive on B ETSELB Transmit on B, receive on A ETPINGPONG Ping Pong mode Return value is less than zero if the function failed. <i>See Appendix B for error codes.</i>
Comments	

ETGetSwitch

Description	Reads the front panel settings of the attached ET-1000 and returns the settings.
Syntax	int ETGetSwitch(SwitchStructure* SStruct)
Parameters	<i>SStruct</i> SwitchStructure* Points to a SwitchStructure structure that is to be loaded with information pertaining to the attached ET-1000's front panel switch settings.
Return Value	Return value is ≥ 0 if the function executed successfully. Return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	See the definition of SwitchStructure below.

SwitchStructure

Parameter	Description
unsigned long Gap	Current Gap Switch setting.
unsigned long Data	Current Data Switch setting.
unsigned Disp	Current Disp Switch setting.
unsigned Mode	Current Mode Switch setting.
int Run	Current Run Switch setting: Run = ETRUN When system is in RUN state. Run = ETSTEP When system is the STEP state. Run = ETSTOP When system is in STOP state.
int Sel	Current Sel Switch setting: Sel = ETSELA When transmitting out Port A. Sel = ETSELB When transmitting out Port B. Sel = ETPINGPONG When the system is in the "Ping Pong" mode.

ETGetTransmitTrigger

Description	Returns the transmit trigger parameters currently implemented by the attached ET-1000.
Syntax	int ETGetTransmitTrigger(TriggerStructure* TStruct)
Parameters	<i>TStruct</i> TriggerStructure* Points to a TriggerStructure structure which is to contain the trigger parameters
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code less than zero is returned if this function failed to execute. <i>See Appendix B for error codes.</i>
Comments	See the definition of TriggerStructure on page 213.

ETGetVFDRun

Description	Returns the current run state of the VFD pattern on the attached ET-1000.
Syntax	int ETGetVFDRun(void)
Parameters	None
Return Value	Return value depends on the VFD run state: ET_OFF VFD NOT being transmitted ET_ON VFD being transmitted A failure code, which is less than zero, is returned if this function failed to execute. <i>See Appendix B for error codes.</i>
Comments	None

ETLNM

Description	Activates or de-activates Live Network Mode.
Syntax	int ETLNM(int Type)
Parameters	<i>Type</i> int Determines the state of the live network mode: ETLNM_ON Live Network Mode ON ETLNM_OFF Live Network Mode OFF
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if this function failed to execute. <i>See Appendix B for error codes.</i>
Comments	Network Mode is currently available only on Port A of the attached ET-1000.

ETLoopback

Description	Activates or de-activates internal loopback of the specified port.
Syntax	int ETLoopback(int Port, int Status)
Parameters	<i>Port</i> int Determines the ET-1000 port for activating or deactivating internal loopback: LOOP_PORT_A Loopback on Port A LOOP_PORT_B Loopback on Port B Any other values are invalid and will have no effect on the attached ET-1000. <i>Status</i> int Determines the loopback status of <i>Port</i> : ETLOOPBACK_ON Loopback the port ETLOOPBACK_OFF Do not loopback the port
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if this function failed to execute. <i>See Appendix B for error codes.</i>
Comments	None

ETMFCounter

Description	Establishes the item to be counted by the associated Multi-Function counter.	
Syntax	int ETMFCounter(int Port, int Mode)	
Parameters	<p><i>Port</i></p> <p>int Determines the ET-1000 port whose associated Multi-Function counter is to be re-assigned:</p> <p>MFPORT_A ET-1000 Port A</p> <p>MFPORT_B ET-1000 Port B</p> <p>All other values are invalid and will not have any effect on the ET-1000.</p> <p><i>Mode</i></p> <p>int Identifies the item to be counted by the Port's Multi-Function counter. Values are:</p> <p>ETMF_PACKET_LENGTH Packet Length</p> <p>ETMF_RXTRIG_COUNT Receive Trigger Count</p> <p>ETMF_TXTRIB_COUNT Transmit Trigger Count</p> <p>ETMF_TIME_ROUNDTRIP Time from Port to Port</p> <p>ETMF_TIME_PORT2PORT Time from Port to other Port</p> <p>ETMF_RXTRIG_RATE Receive Trigger Rate</p> <p>ETMF_TXTRIG_RATE Transmit Trigger Rate</p> <p>ETMF_PREAMBLE_COUNT Number of preamble bits in Port</p> <p>ETMF_GAP_TIME Packet Gap Time in Port</p> <p>ETMF_SQE_COUNT SQE count in Port</p> <p>ETMF_TOTAL_LENGTH Total packet length in Port</p> <p>All other values are invalid and will not have any effect on the ET-1000.</p>	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>	
Comments	None	

ETPreamble

Description	Sets the preamble bit count that is to be transmitted by the attached ET-1000.	
Syntax	int ETPreamble(int Count)	
Parameters	<p><i>Count</i></p> <p>int Specifies the number of preamble bits to be inserted into the transmit stream of the attached ET-1000.</p> <p>Range: 10 to 128. Values outside this range are invalid and will have no effect on the attached ET-1000.</p>	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>	
Comments	None	

ETReceiveTrigger

Description	Sets up the receive trigger on the attached ET-1000.
Syntax	int ETReceiveTrigger(TriggerStructure* RStruct)
Parameters	<i>RStruct</i> TriggerStructure* Points to a TriggerStructure structure that contains all the trigger information necessary to set up the receive trigger on the attached ET-1000.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	See the definition of TriggerStructure on page 213.

ETRemote

Description	Sets the attached ET-1000 in either the local or remote mode.
Syntax	unsigned ETRemote(int Mode)
Parameters	<i>Mode</i> int Determines the mode in which the attached ET-1000 operates: ETLOCALMODE Local Mode ETREMOTEMODE Remote Mode All other values are invalid and will have no effect on the attached ET-1000.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Once the attached ET-1000 is placed in the local mode, it will no longer respond to instructions sent to it by the PC—except, of course, the instruction generated by ETRemote. This function will typically be used to place the attached ET-1000 in local mode so that it responds to user input from its front panel. (In remote mode, all front panel functions except DISPLAY and RESET are inoperative.)

ETReset

Description	Resets all counters on the attached ET-1000.
Syntax	int ETReset(void)
Parameters	None
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This function essentially emulates the activation of the attached ET-1000's front panel RESET switch.

ETReturnAddress

Description	Returns the same void pointer passed.
Syntax	void * ETReturnAddress(void *)
Parameters	<i>p</i> void* Standard pointer.
Return Value	avoid * (32 bit value, which in Visual Basic is a long)
Comments	Visual Basic does not have a pointer type, yet can pass arguments by reference. The HTVFD structure includes a pointer. This function is a workaround to allow a long to be used as a pointer for the HTVFDStructure. This is seen in the example snippet in the VFD bug fix above.

ETRun

Description	Sets the run state on the attached ET-1000.
Syntax	int ETRun(int RunValue)
Parameters	<p><i>RunValue</i> int Determines the run state to be executed on the attached ET-1000:</p> <p>ETSTOP Halts transmission</p> <p>ETSTEP Sends a single packet</p> <p>ETRUN Sends continuous packets</p> <p>All other values are invalid and have no effect on the attached ET-1000.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	The result of executing this function differs somewhat when the attached ET-1000 is in the BURST mode. See the ETBurst function.

ETSetJET210Mode

Description	To set up the attached ET-1000 to operate with or without a JET-210 (Jitter Simulator) attached.
Syntax	int ETSetJET210Mode(int Mode)
Parameters	<p><i>Mode</i> int Sets the JET-210 mode of the attached ET-1000:</p> <p>ET_OFF Disable the JET-210 mode</p> <p>ET_ON Enable the JET-210 mode</p> <p>All other values are invalid and will not work on ET-1000.</p>
Return Value	The return value is >= 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Since the JET-210 Jitter Simulator assumes control over the three rear panel BNC connectors on the attached ET-1000, the BNC functions will be pre-empted. Use the ETBNC function to re-establish BNC functionality after disabling the JET-210 mode. Disabling the JET-210 mode with this function effectively puts the three BNC connectors into Input mode.

ETSetSel

Description	Determines the transmission function associated with Port A and Port B of the attached ET-1000.
Syntax	int ETSetSel(int SelValue)
Parameters	<p><i>SelValue</i> int Determines mode associated with the ET-1000 ports:</p> <p>ETSELA Transmit on A, receive on B</p> <p>ETSELB Transmit on B, receive on A</p> <p>ETPINGPONG “Ping Pong” mode</p> <p style="text-align: center;">All other values are invalid and will not work on the ET-1000</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

ETSetup

Description	Stores and recalls the current setup internally in the attached ET-1000.
Syntax	int ETSetup(int Mode, int SetupId)
Parameters	<p><i>Mode</i> int Determines the mode of the setup function:</p> <p>ETSTORESETUP store the current setup</p> <p>ETRECALLSETUP recall a stored setup</p> <p style="text-align: center;">All other values are invalid and will have no effect on the ET-1000.</p> <p><i>SetupId</i> int Identifies the specific setup to store or recall. For recall, this value ranges from 0 to 8; whereas 0 is the “factory default” setup. (It cannot be changed.) You are allowed to store setups 1 to 8. Any values outside these ranges are invalid and will have no effect on the ET-1000.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	<p>The setups referenced in this function refer to setups that are stored internally within the attached ET-1000. There are no library functions available for storing and recalling setups from the PC’s disk.</p> <p>NOTE: Recalling a previous setup in the ET-1000 will probably result in the loss of the communication link. After executing this function, your application program should unlink itself from the attached ET-1000 and then re-link. Use the following procedure:</p> <ol style="list-style-type: none"> 1. Issue the ETUnLink command 2. Wait 4 seconds. This allows the ET-1000’s serial port to settle after the recall operation. 3. Re-link using the ETLink(...) function. <p>You may find that a re-link will result in a different Baud rate than before. Use the ETSetBaud(...) function if you wish to re-establish the link at a particular Baud rate. (Note that after issuing ETSetBaud, you must again UnLink and then Link.)</p>

ETTransmitCRC

Description	Enables or disables transmission of CRC errors on the attached ET-1000.
Syntax	int ETTransmitCRC(int Active)
Parameters	<p><i>Active</i> int Determines the state of the CRC error insertion on the attached ET-1000:</p> <p>ETCRC_ON Enable CRC transmission</p> <p>ETCRC_OFF Disable CRC transmission</p> <p>All other values are invalid and have no effect on the attached ET-1000.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

ETTransmitTrigger

Description	Sets up the transmit trigger on the attached ET-1000.
Syntax	int ETTransmitTrigger(TriggerStructure* TStruct)
Parameters	<p><i>TStruct</i> TriggerStructure* Points to a TriggerStructure structure that contains all the trigger information necessary to set up the transmit trigger on the attached ET-1000.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	See the definition of TriggerStructure on page 213.

ETVFDParams

Description	Sends VFD information to the attached ET-1000.
Syntax	int ETVFDParams(VFDStruct* VFDdata)
Parameters	<p><i>VFDdata</i> VFDStruct* Points to a VFDStruct structure which contains all the VFD information required to implement a VFD pattern on the attached ET-1000.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Depending on the size of the Range parameter of <i>VFDdata</i> , this function may take some time to download its information to the attached ET-1000. See the definition of VFDStruct below.

VFDStruct

Parameter	Description
unsigned Offset	Specifies the position in the transmit data stream where VFD data begins. Measured in bit times elapsed since the final preamble bit. <i>Range: 0 to 65535 (0x0000 to 0xFFFF).</i>
unsigned Range	Specifies the size of the VFD word in bytes. <i>Range: 1 to 4095(0x0001 to 0xFFFF).</i>
int Start [4096]	Contains the VFD Start pattern. Start[0] is the LSByte, Start[4095] is the MSByte.
int Increment [4096]	Contains the VFD Increment (decrement) word. Increment[0] is the LSByte, Increment[4095] is the MSByte. Because of this structure's large memory requirements, it is recommended that you dynamically allocate and deallocate memory for it. See the example below.

Example for int Increment[4096] in VFDStruct

```

main()
{
VFDStructure *VFD;      //pointer to a VFD structure
VFD = (VFDStructure*)malloc(sizeof(VFDStructure));
//allocates memory
VFD->Range = 32;
VFD->Offset = 8;        //for example:
                        //code to set up the data patterns
ETVFD(VFD);            //send to ET1000/SMB-1000
{}                      // other code...
free(VFD);             //deallocates far memory
}

```

ETVFDRun

Description	This function starts or halts the transmission of VFD data from the attached ET-1000.
Syntax	int ETVFDRun(int Start)
Parameters	<p>Start int Determines the state of the VFD transmission:</p> <p>ETVFD_ENABLE Enable VFD transmission</p> <p>ETVFD_DISABLE Disable VFD transmission</p> <p>All other values are invalid and have no effect on the attached ET-1000.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	VFD information must first be sent to the attached ET-1000 using the ETVFDParams function. Once the ETVFDParams function has set up the VFD parameters, VFD transmission may be enabled and disabled numerous times without the need to execute ETVFDParams again—as long as the VFD data doesn't need to be changed. If ETVFDParams is not executed before this function, the attached ET-1000 will implement whatever VFD information it contains. NOTE: Sometimes the ET-1000 will power-up with VFD active and running. Use ETGetVFDRun to determine whether or not this is so, and then use ETVFDRun(...) to place the ET-1000 in a known state.

Appendix B:

Error Codes

When a function is not successful, an error code is returned instead of data. Error code values are always less than zero. They may be signed integers or signed long integers.

Error Value	Definition	Description
-1	UNSPECIFIED ERROR	An error condition was encountered but could not be identified. This will occur if the system experienced an error that does not fit into any of the above categories.
-2	PORT_NOT_LINKED	An attempt to use a Programming Library function was made without an active link to the SmartBits.
-3	UNLINK_FAILED	An attempt to unlink the SmartBits from the serial port failed. This could occur if the SmartBits is already unlinked from the port before the ETUnLink command is called.
-4	INCORRECT_MODE	The attached SmartBits was put into a mode of operation where the attempted call to the library function was not applicable. For instance, you cannot access packet data unless the capture mode has been enabled.
-5	PARAMETER_RANGE	An incorrect or invalid range was specified for a library function parameter. This may include ranges within structures whose pointers are passed as a parameter to the function.
-6	PACKET_NOT_AVAILABLE	An attempt was made to access information from an indexed packet not currently in the capture buffer of the attached SmartBits.
-7	SERIAL_PORT_DATA	No errors were detected on the serial port, but the data returned from it doesn't appear to be correct. This indicates a serial port with interference. Try reducing the baud rate by modifying the ETSetBaud() parameter.
-8	ET1000_OUT_OF_SYNC	The attached SmartBits is operating in a mode different from what was expected. Perform an ETUnLink command followed by Link.
-9	PACKET_NOT_FOUND	An attempt was made to locate a packet within the SmartBits's capture buffer, but the packet contents could not be found and/or verified.
-10	FUNCTION_ABORT	The user aborted a function before it could run to completion.
-11	ACTIVE_HUB_NOT_INITIALIZED	An attempt to execute a command that requires a card was unsuccessful because the library failed to properly initialize the board. The library will always try to initialize the board if it hasn't been done so already, but for some reason, the initialization failed. This could indicate a failed card.
-12	ACTIVE_HUB_NOT_PRESENT	An attempt to execute a command that requires a card was unsuccessful because the addressed port had no board installed in it.
-13	WRONG_HUB_CARD_TYPE	An attempt to execute a command that requires a card was unsuccessful because the addressed port contained a Passive Hub board.

Appendix B
Error Codes

Error Value	Definition	Description
-14	MEMORY_ALLOCATION_ERROR	An attempt to execute a command that requires a card was unsuccessful because the addressed port contained a Passive Hub board.
-15	UNSUPPORTED_INTERLEAVE	Not currently implemented.
-16	PORT_ALREADY_LINKED	The Programming Library supports one connection at a time to a SmartBits. An ETLINK command was issued when an active link already exists.
-17	HUB_SLOT_PORT_UNAVAILABLE	A request was made to perform an operation on a Hub/Slot/Port that does not exist in the current configuration.
-18	GROUP_HUB_SLOT_PORT_ERROR	A request was made to create or perform an operation on a group with a Hub/Slot/Port that does not exist in the current configuration.
-19	REENTRANT_ERROR	An attempt was made to call a Programming Library function while BackgroundProcessing was enabled, and the Programming Library was already performing a function.
-20	DEVICE_NOT_FOUND_ERROR	An attempt was made to address an attached device that could not be found [e.g. an MII transceiver].
-21	PORT_RELINK_REQUIRED	The connection is down, but no disconnect action was taken by either side.
-22	DEVICE_NOT_READY	Current use: Token Ring is down.
-23	GROUP_NOT_HOMOGENEOUS	Not currently implemented. (Used only by undocumented commands).
-24	INVALID_GROUP_COMMAND	Not currently implemented. (Used only by undocumented commands).
-25	ERROR_SMARTCARD_INIT_FAILED	Unable to initialize card.
-26	SOCKET_FAILED	Error in the socket connection for an Ethernet Link (PC to SMB).
-27	SOCKET_TIMEOUT	Timeout on the socket connection for an Ethernet Link (PC to SMB).
-28	COMMAND_RESPONSE_ERROR	Invalid command response received from SmartBits.
-29	CRC_ERROR	CRC error in the data transfer.
-30	INVALID_LINK_PORT_TYPE	An attempt was made to link a PC to a SmartBits chassis over a connection that is recognized as neither a normal Serial Comm Port nor a proper TCP/IP Socket Link. (This error message should not occur.)
-31	INVALID_SYNC_CONFIGURATION	User attempted to perform a GPS sync action when the SmartBits is not set for GPS. (Could indicate that GPS is not ready.)
-32	HIGH_DENSITY_CONTROLLER_ERROR	
-33	HIGH_DENSITY_CARD_ERROR	
-34	DATA_NOT_AVAILABLE	An attempt was made to retrieve data from a card when no data of the intended type was available. An example would be when you attempt to retrieve histogram results from a Layer3 card when no histogram information has been accumulated yet.
-35	UNSUPPORTED_PLATFORM	The function is not available on the current platform. Currently this is used for HTDefaultStructure and related functions that are not available on 16-bit Windows platforms.

Error Value	Definition	Description
-36	FILE_IO_ERROR	An error occurred in accessing a file. Currently this will occur when HTDefaultStructure or a related function is called and the defaults file is not found.
-37	MULTI_USER_CONFLICT	The attempted action conflicted with another user of the same SmartBits. This will occur when a GPS sync start or stop is attempted shortly after another user also attempts a GPS sync action.
-98	SERIAL_PORT_TIMEOUT	The serial port timed out while waiting for a response from the SmartBits. This usually indicates a problem with the physical serial link.
-501	NSTCL_PARAMETER_TYPE	This error will occur when a given parameter is not of the expected type, for example if a non-numeric argument is given when an integer is expected. This error often occurs when a "\$" is missing in front of a variable name.
-502	NSTCL_INVALID_MSG_FUNC	An attempt to unlink the SmartBits from the serial port failed. The Tcl interface is unable to process a message function because it doesn't recognize the given iType parameters as matching the accompanying data structure for the card at the specified location.

Appendix C:

Library Revision History

Version 3.07

Version 3.07 includes all changes made after the official 3.05 SmartLib release, including improvements made in:

- SmartLib 3.05 patch 1, patch 2, patch 3
- SmartLib 3.06 beta and beta 2
- SmartLib 3.07

Major Enhancements

- Multiuser support for the SMB-6000, SMB-600, and multiuser-capable SMB-2000.
- Support for several new SmartCards, including LAN-6100, POS-6500, LAN-6201, and ML-5710 cards.
- Default values provided for message functions.
- Greatly simplified Tcl interface.

APIs Still Under Construction

Although SmartLib 3.07 Original Commands and Message Functions are ready for use and include full technical support, SmartAPI for SmartApplications and SmartAPI for SmartSignaling are still in progress and are not ready for use in this release.

To use either of these APIs, use SmartLib version 3.05 or earlier.

Win16 Support Ending Soon

SmartLib support for the Win16 platform (Windows 3.1, Windows for Workgroups 3.11, etc.) will end after SmartLib 3.07. Future releases will not have support for the Win16 platform.

New Features and Fixes

- HTDefaultStructure function added to provide default values for message function structures. The defaults file is in a user-editable ASCII format.
- The new Tcl interface, called smartlib.tcl, provides these benefits:
 - ❖ Using defaults from Tcl has been made easier by providing defaults automatically in the HTSetStructure function.
 - ❖ The need for format commands for values being passed to SmartLib has been eliminated
 - ❖ Setting arrays and array elements in Tcl has been made significantly easier by removing previously required syntax overhead
 - ❖ The library now accepts single array elements as function parameters.
- Support for the POS-6500 SmartCard
- Support for the LAN-6201 SmartCard

- Support for the LAN-6100 SmartCard
- Support for the ML-5710 SmartCard
- Frame protocol filling functions now support TCP, UDP, and IP over Ethernet and 2 ATM encapsulations (ENCAP_ATM_PVC and ENCAP_ATM_SVC_SNAP)
- Support for multiple simultaneous connections to all multiuser-capable chassis models. This includes the SMB-6000, the SMB 600, and multiuser-capable SMB-2000's. New functions were added, including NSSocketLink, HTSlotReserve, HTSlotRelease, and HTSlotOwnership. NSSocketLink is similar to ETSocketLink with the additional capability to control the reservation of ports at link time. HTSlotReserve allows a specific slot to be reserved exclusively for the current user. HTSlotRelease allows a previously reserved card to be released for use by others. HTSlotOwnership provides for slot reservation checks.
- Native port mapping for the SMB-6000 and SMB-600. Native mapping mode allows individual ports in slots to be addressed using their actual port number. See the user manual for more details.
- Added a new error code, MULTI_USER_CONFLICT returned when a synchronized start is attempted while another user is using the GPS subsystem
- ATM Message functions providing enhanced stream indexes are now available for use.
- ATM-9155 A and B cards can now be distinguished from ATM-9155 C cards via a new constant returned from HTGetCardModel
- The ability to enable or disable inverse ARPs on ATM cards was added via the ATMClassicalIP structure's new ucInvArpReplyOff field
- PPP support for the POS, WAN and ATM SmartCards.
- The NSSetControllerID command was added to allow the configuration of the controllerID for SmartMetrics tests on high-density chassis
- Changed message returned by sourcing et1000.tcl
- New error codes have been added. -34 represents "data not available" during communications with the high-density controller family. -35 (UNSUPPORTED_PLATFORM) is for functions that are not available on the given operating system. FILE_IO_ERROR (-36) means that a file couldn't be read or written. Usually it means the default value file cannot be found. MULTI_USER_CONFLICT (-37) is returned in the case when two users attempt an operation requiring a shared resource, like an attached GPS unit, at the same time.
- Error codes -501 and -502 are returned by the Tcl interface when it detects invalid parameters or message functions, respectively.
- Visual Basic and Delphi interfaces have been fixed to resolve minor errors
- Bug #5359: ETH_TRANSMIT/LAN-6100A: SmartLib does not pass the value assigned to uiVFD3DataCount to the SmartCard. Instead it passes the value defined in uiVFD3BlockCount. Currently the LAN-6100A does not support VFD Block Counts.
- Bug #5227: With UNIX Library HGSetSpeed and HGDuplexMode do not work unless you access the MII registers with another command or program. Dumping the MII registers shows the commands are not changing the values on the registers. (After you look any subsequent HGSetSpeed or HGDuplexMode commands will work on the port you looked at.
- Bug #5146: HGGetCounters doesn't work with multiple socket links. It gets valid counter values from the first socket link only; counter values from the rest of the links are 0. Test script: project/tests/sync8.tcl

- Bug #4989: GPS synchronized starts were delayed by an hour after the daylight savings time changeover.
- Bug #4925: HTGetStructureSize returning zero for FST_PROTOCOL_COUNTERS_INFO, and FST_PROTOCOL_PARAMS
- Bug #4863/4864: NSSetPayload has been fixed for a couple of cases where it wasn't setting the fill pattern as requested.
- Bug #4831: NSCreateFrame problem: When setting the ipattern=PAT_DECW, and using HTFrame to load into the cards, the frame which is just created would be like FF 00 FF 00 FF 00 FF 00. This problem happens on both ML-7710 and SX-7410.
- Bug #4830: NSCreateFrame problem: When setting the ipattern=PAT_INCW, and using HTFrame to load into the cards, the frame which is just created would be like 00 00 00 02 00 04 00 06. This problem happens on both ML-7710 and SX-7410.
- Bug #4676: ETHTransmit structure ucAlignErrors field documentation fixed to properly indicate that the field is a count of bits instead of simply a boolean value.
- Bug #4632: ETHCounterInfo's ulRxFrameRate and ulTxFrameRate fields were transposed.
- Bug #4622: UNIX install script asks if you want to install Tcl 8.0 and implies that answering no will install 7.6 instead. Has been clarified to remove the confusing implication.
- Bug #4212/4211: ETGetLinkStatus return value for an Ethernet link was indistinguishable from COM3. It has been fixed so that an Ethernet link returns 32000, distinguishing it from serial links (though not from other ethernet links)
- HTGetHubLEDs fixed to return an error code when used on a 6000-family chassis. Previously it would behave unpredictably
- Fixed problem with linking and unlinking. If one link failed, subsequent link and unlink attempts returned erroneous values.
- Fixed ETGetController to return the appropriate ET-1000 constant for an ET-1000.
- Fixed problem unlinking with multiple active links.
- Fixed bug with ETH_TRIGGER message function where trigger two was not being properly set.
- Fixed byte offset problem in ETH_TRANSMIT message function with triggers and VFDs of fewer than 6 bytes.
- Fixed problem where log output would sometimes show up as ASCII instead of binary data.

Version 3.06

- Beta release only. Version 3.06 additions and changes are included in the 3.07 description above.

Version 3.05

New Features and Fixes

- Per-Connection Burst Count (ATM-1 and ATM-2). This feature enables applications controlling the ATM cards to specify a quantity of frames to transmit (and then stop) for each active connection.

- Per-Port Burst Count (ATM-1 and ATM-2). This feature is the same as above, except here we specify it for each card or per-port basis inclusive of all active connections).
- Added support for the SMB-6000 SmartBits chassis and the LAN- 6200A SmartCard, to the level of compatibility with the SMB-2000 SmartBits chassis and the GX-1405 SmartCard.
- New function: ETSetGPSDelay(unsigned long ulSeconds); to set the delay time before a GPS synchronized start/stop.
- Fixed RemoveHubSlotPortFromGroup(), which only worked in certain cases. Now it should work all the time.
- Fixed bug in HTDuplexMode to allow half-duplex settings.
- Fixed bug in one-to-many test if ATM is on one side (Bug #3920).
- Changed HTBurst “AH” for “mode” command.
- Fixed bug for throughput test. Rate never increased when ATM card was the source.
- One-to-many ATM test improved to support result retrieval when multiple streams have only one connection.
- Fixed bug where ATM cards after first card weren’t being initialized in one-to-many or many-to-one tests.
- Fixed bug where ATM card was being initialized several times.
- Added function ETIsSyncCapable for GPS support.
- Added one more decimal place of resolution to status results.
- Changed ucSearchType field to ulSearchType.
- Added utility functions for U64 structure
- Fixed SASA bug in ARP replies: IP destination was incorrect when both “multiple trials” and “learning every trials” options elected.
- Report format modified to allow apps to create tabular reports.
- SASA corrected to check packet errors before calculating throughput results.
- Store and Forward latency calculation fixed for Token Ring, 100Mbit Ethernet, and 1Gbit Ethernet.
- Extended frame relay timeout period to be $2 * NN1 * NT1$ to fix a reported bug.
- Fixed problem with decoding 2 bytes of the lecid returned back from card.
- Per-Connection Burst Count (ATM-2). This feature enables applications controlling the ATM cards to specify a quantity of frames to transmit (and then stop) for each active connection.
- Per-Port Burst Count (ATM-2) This feature is the same as above except here we specify it for each card or per-port basis inclusive of all active connections)
- Cell Scheduling (ATM-2) This feature provides the ability to schedule N connections equally and at a specific percentage of line rate.
- Stream Copy, Stream Modify and Stream Fill for ATM cards. This feature helps reduce the setup time associated with configuring streams. Stream Copy creates a given number of streams (up to the max for the card) that are identical to an already existing “source” stream. Stream Modify modifies the parameters of a given number of streams that already exist on a

card, with an absolute value. Stream Fill, is similar to the Stream Modify feature, except here a delta value to increment, from the initial value, is specified.

- Frame Copy for ATM cards. This feature helps reduce the setup time with configuring frames. Frame Copy defines frames for multiple existing streams in a single command.
- Histogram retrieval from the frame relay cards has been fixed. Index and count now work as well for the FR_HIST_LATENCY_INFO iType.
- Modified HTSeparateHubCommands(HUB_GROUP_SYNC_ACTION) to return error if SmartBits is not configured properly for GPS or synchronized start.
- Modified Tcl interface to allow either of the following syntaxes for declaring a single element array: “struct_new ulVar ULong” OR “struct_new ulVar ULong*1”.
- Added SMB-200 support for ETGetController() function—returns CONTROLLER_SMB200 constant.
- HGSetGroup fixed so it can support setting a group across multiple links. (Bug #3767)
- ETLink, ETSocketLink fixed so that if it is successfully executed, the return value will be the new link count.
- Added the capability to change gigabit latency adjustment factors from the .ini file.
- Modified ETLink to check if a comport is already linked before attempting to link again. (Bug #3894)
- Changed FieldCount member of Layer3ModifyStreamDelta to FieldRepeat.
- Fixed problems with HGResetPort.
- Added capability to specify which histogram records to retrieve.
- Added new error codes.
- Fixed report file and log file problem for Unix (Bug #4278)
- New constant names for HTSeparateHubCommands.
- Modified ETSetTimeout to use max timeout when given 0 as the timeout parameter.
- Fixed HTLatency to not require background pattern to be set separately.
- Fixed Unix byte-ordering problems with SmartAPI.

Version 3.04

New Features and Fixes

- Added Ethernet message functions.
- Support for starts synchronized by GPS added.
- Added support for ATMClassicalIPInfo structure.
- Added support for T1/E1 Frame Relay cards.
- Fixed bug where StopOnError failed to stop test under UNIX SmartAPI for SmartApps default error callback.
- Fixed ETGetBaud bug with multiple links.
- Fixed HGSetSpeed function.
- Added Tcl and C sample code.

- Fixed timeouts on high-latency connections.
- Added delay for UNI restart on ATM cards.
- Added option to set up Stream8023 streams for Frame Relay cards in the SmartAPI for SmartApps.
- Increased latency resolution from 32 to 64 bits with ATM cards.
- Added support for Gigabit autonegotiation to SmartAPI for SmartApps.
- Changed SmartAPI for SmartApps to allow back-to-back test to reach 100% regardless of resolution setting.
- Changed SmartAPI for SmartApps to report packet loss based on the transmitter rate instead of the receiver.
- Added IUseIdenticalRate parameter to ATM setup for SmartAPI for SmartApps.
- Added uiMaxRateWithTeardown and uiMaxRateWithoutTeardown into ATMCardCapability structure.
- Changed HTResetPort to stop ping, SNMP, and ARP reply packets from being transmitted from Layer 3 cards.
- Fixed bug in HTGetStructure when used with ATM cards to retrieve more than 2048 bytes of data.
- Added VFD1, 2, and 3 Block count to support 7710.
- Added support for WN-3415 and -3420 to HTGetCardModel.
- Added commands for ATM Classical IP client establish/release
- Fixed bug causing L3 and ML cards to crash if reset while running.
- Fixed bug with WriteMII to register 0.
- Added new command for per-connection burst count.
- Added support for UNI 3.0 signaling in the back to back mode for the SmartSignaling API.
- Modified the test approach for the Call Capacity test of the SmartSignaling API. The test will now run until all connections have failed rather than quitting after the first failed connection.
- The timestamps in the Signaling API are now 64 bits long supporting time durations to 58,000+ years.
- Added HGClearGroup command to replace obscure HGSetGroup(NULL)
- Added “Frame” functions for easy static frame generation. Functions allow multiple frames to be created, modified, and set as the fill pattern. Sensible default frame values are placed into new frames, and the CRC is recalculated automatically as the frame contents are altered.
- Fixed installation problems under SunOS 4.1.4. The installation is now successful with the following items installed first: GCC shared library, GNU Make 3.77, and GNU ld 2.9.1 (which comes in GNU binutils 2.9.1).
- Programming Library extension to Tcl 7.6 or 8.0 now installs successfully under SunOS 4.1.4.
- Fixed excessively long timeout for duplicate socket link.
- Fixed excessively long timeout for unlink from dead SmartBits.
- Added embedded structure definitions in Message Functions manual.
- Corrected code omissions in SmartAPI Manual.

- Split SmartAPI manual into: SmartAPI for SmartApplications and SmartAPI for SmartSignaling.
- Manuals converted to full-size 8.5 X 11-inch page format.
- Extensive documentation about histograms (SmartMetrics Results).

Version 3.03

New Features and Fixes

- Added Frame Relay SmartCard support to TestAPI.
- Implemented HTResetPort and HGResetPort for Gigabit SmartCards.
- Added Enable Pause Flow Control option to TestAPI for Gigabit and Fast Ethernet SmartCards.
- Added synchronized start capability between master and slave links.
- Support for Gigabit SmartCard VFD3 buffer sizes of up to 16K.
- Fixed minor Gigabit SmartCard VFD3 bugs.
- Added interface support for Tcl 8.0 to Windows SmartLib.
- Extended maximum number of calls for ATM SmartCards from 512 to 8388607.
- Added Linear Search for the ATM Peak Call Rate test in the TestAPI.
- Added option of no call teardowns for ATM Peak Call Rate test (affecting *Message Functions* and *SmartAPI for SmartSignaling*).
- Additional Smart API result format.
- Other miscellaneous minor bug fixes and improvements. Contact Technical Support for complete list.

Documentation

- New manual, SmartLib Smart API, covering functionality and concepts.
- Corrected examples in SmartLib User Guide.
- Miscellaneous updates and corrections.

Version 3.02

New Features and Fixes

Added support for the following SmartCards:

- ML-7710 100Mb Multi-Layer 10/100 Mbps Ethernet SmartCard
- Reworked all gap commands to send all data in nanoseconds to be consistent with SmartWindow
- Increased receive time-out for command downloads
- Fix of capture count retrieval
- Added Frame relay Get_Structure call to return WAN card version
- Corrected static ILMI command
- Full list available from Technical support

- Corrected report file results problem
- Corrected GbE gap size update

Documentation

- *SmartLib User Guide*. Major update for new functions supported in Version 3.00 and 3.02.
- *SmartLib Message Functions* manual. New manual. Used with the *SmartLib User Guide*, it covers the newer SmartLib Hardware API functions in detail. It contains a complete list of the SmartLib 3.02 message functions and all related parameters. It includes basic concepts of the message function syntax, as well as examples specific to different programming languages.

Version 3.00

New Features and Fixes

Added support for the following SmartCards:

- SX-7410 100Mb Fast Ethernet
- AT-9622 622Mb OC-12c ATM
- AT-9155 155Mb OC-3 ATM Signaling and Frame Generation
- AT-9045 45Mb DS3 ATM Signaling and Frame Generation
- AT-9034 34Mb E3 ATM Signaling and Frame Generation
- AT-9020 2.048 E1 ATM Signaling and Frame Generation
- AT-9015 1.544 T1 ATM Signaling and Frame Generation
- GX-1405 Gigabit Ethernet
- WN-3405 V.35 Frame Relay

Visual Basic Interface Changes

- Added updated Visual Basic Interface files. These are in the VB directory with filenames matching their corresponding .h header files. The 16-bit VB files have extensions .b16. The 32-bit VB files have extensions .b32. The new VB interface files contain updated commands, structures, and constants. They also include the following changes from the previously distributed files:
 - HTVFDSstructure: iPointer and iLength fields have been renamed to pData and DataCount respectively, to more closely match the field names in et1000.h.
 - The previously distributed VB interface files (etsmbapi.txt, etsmbw32.txt, and atmitem32.txt) are still distributed in the CommLib directory, for use with previously written tests. They do not contain updated commands, structures, and constants.

Version 2.50-20

New Features and Fixes

- Added TestAPI functions to perform the RFC1242 tests and retrieve test results. New functions include:
 - ❖ int NS1242TestStart(int iTestType
 - ❖ PortPairStruct *pPortPair

- ❖ int iTestPairs
- ❖ TestSetup *pTestSetup
- ❖ StatusCallbackFunc StatusCallBack
- ❖ ErrorCallbackFunc ErrorCallBack)
- ❖ int NS1242TestStartVB(int iTestType
- ❖ PortPairStruct *pPortPair
- ❖ int iTestPairs
- ❖ TestSetup *pTestSetup)
- ❖ int NS1242TestStop(int iTestType)
- ❖ int NS1242TestReport(int iTestType, char *pszReport)

Version 2.42

New Features and Fixes

- Added functions to set and save card speed and duplex modes.
- Added functions to get the card specific minimum and maximum interpacket gap allowed and acceptable, and length allowed and acceptable.

Version 2.37

New Features and Fixes

- Added functions to save trigger configurations.
- Fixed bug where port 79 (hub 4, port 19) card type was being overwritten.
- Fixed Interburst Gap.
- Added HGStartSetGroup and HxModifyFillPattern.
- Fixed VB prototypes
- Automatically defer sending group configure hub group command until group start/stop/step is required. This can result in very large speedups when using HGSetGroup and HGAddToGroup in a loop.
- Added the STATUS_XXX items that are documented under the GetEnhancedStatus() manual. However, entered the values as the correct values being returned from the TokenRing card.
- In HTHubSlotPorts(), added valid returns for CT_TOKENRING and CT_VG.

Version 2.32

New Features and Fixes

- Fixed behavior for Multiburst gap for 100mb cards.
- Added optimization for HGAddToGroup command where an HGStartSetGroup()/HGEndSetGroup pair can bracket a multiple change of ports in a group to speed up command processing time.

- Added HGModifyFillPattern and HTModifyFillPattern to allow multiple cards to be programmed followed by a difference file for particular cards.

Version 2.31

New Features and Fixes

Added library commands for VG SmartCard:

- int HGSetVGProperty(pVGPStructure)
- int HTSetVGProperty(pVGPStructure, iHub, iSlot, iPort)

Version 2.3

New Features and Fixes

Added library commands for better group configuration control:

- int HGGetGroupCount(void)
- int HGRemoveFromGroup(int iHub, int iSlot, int iPort)
- int HGRemovePortIdFromGroup(int iPortId)
- int HGIsPortInGroup(int iPortId)
- int HGIsHubSlotPortInGroup(int iHub, int iSlot, int iPort)

Added TokenRing SmartCard commands:

- int HTPortProperty(unsigned long* pulProperties,int iHub, int iSlot, int iPort)
- int HTSetTokenRingErrors(iTRErrors, iHub, iSlot, iPort)
- int HTSetTokenRingAdvancedControl(pTRAdvancedStructure, iHub, iSlot, iPort)
- int HGSetTokenRingAdvancedControl(pTRAdvancedStructure)
- int HGSetTokenRingErrors(iTRErrors)
- int HTSetTokenRingProperty(pTRPStructure, iHub, iSlot, iPort)
- int HTSetTokenRingLLC(pTRLStructure, iHub, iSlot, iPort)
- int HTSetTokenRingMAC(pTRMStructure, iHub, iSlot, iPort)
- int HTSetTokenRingSrcRouteAddr(UseSRA, piData, iHub, iSlot, iPort)
- int HTGetEnhancedCounters(pEnCounter, iHub, iSlot, iPort)
- int HTGetEnhancedStatus(piData, iHub, iSlot, iPort)
- int HGGetEnhancedCounters(pEnCounter)
- int HGSetTokenRingProperty(pTRPStructure)
- int HGSetTokenRingLLC(pTRLStructure)
- int HGSetTokenRingMAC(pTRMStructure)
- int HGSetTokenRingSRA(UseSRA, piData)

Added link status commands. These COM port “linkage” related functions now allow multiple ET-1000 and/or ETSMB-1000 systems to be connected and controlled from a single program using the ETSMB Programming Library.

- int ETSetCurrentLink(ComPort)
- int ETGetCurrentLink()
- int ETGetLinkFromIndex(iLink)
- int ETGetTotalLinks()

Version 2.22

New Features and Fixes

- Fixed Gap scale and gap range problem.
- Documented HTCollisionBackoffAggressiveness().

Version 2.21

New Features and Fixes

- int ETGetLibVersion(pszDescription, pszVersion)
- long ETGetBaud();
- int HTFindMIIAddress(pAddress,pControlBits,hub,slot,port).
- Now allow Range = 0 when HTVFD set to HVFD_NONE.
- Fixed a bug in RecallSettings() when being issued to a 100 Mbps FastCard.

Version 2.20

New Features and Fixes

- Added support for 100 Mbps Fast cards.
- Added HTReadMII and HTWriteMII functions to support the 100 Mbps Fast cards.
- Added HTDuplexMode() and HGDuplexMode().
- The Range for (ET)VFDStructure Base pattern and Increment buffer has been limited to 4096 bytes.
- The packet length may now range from 1 to 8191 bytes in the HTDataLength() command to allow runts and jabbers. A value of zero still generates random lengths.
- Extended the HTVFDStructure.Range member to allow specifying bit sized fields for VFD1 and VFD2.
- Added library commands for the following SmartCard controls:
 - ❖ int HTTransmitMode(iMode, hub, slot, port)
 - ❖ int HTBurstCount(lCount, hub, slot, port)
 - ❖ int HTInterBurstGap(lCount, hub, slot, port)
 - ❖ int HTInterBurstGapAndScale(lCount, iScale, hub, slot, port)
 - ❖ int HTMultiBurstCount(lCount, hub, slot, port)
 - ❖ int HTGapAndScale(lCount, iScale, hub, slot, port)

—and the corresponding hub group commands:

- ❖ `int HGTransmitMode(iMode)`
- ❖ `int HGBurstCount(lCount)`
- ❖ `int HGInterBurstGap(lCount)`
- ❖ `int HGInterBurstGapAndScale(lCount,iScale)`
- ❖ `int HGMultiBurstCount(lCount).`
- ❖ `int HGGapAndScale(lCount, iScale)`
- The two commands, `HxTransmitMode()`, and `HxBurstCount()` replaces the single command `HxBurst()`. The `HxBurst()` command was used to set the burst count, and then immediately set the transmit mode. With the introduction of the `HxTransmitMode()` command, the user now has explicit control over the transmit mode. Future commands should use the `HxTransmitMode(iMode)`, and `HxBurstCount(lCount)` commands and no longer utilize the `HxBurst()` command.
- The introduction of the `HxGapAndScale()` commands affect the interpretation of the `HxGap()` command. Please review the detailed description of each command for specific behaviors in common usage.

Version 2.13

New Features and Fixes

- Added missing `HGSelectTransmit` prototype.
- Fixed sample ET-1000 initialization code.

Version 2.12

New Features and Fixes

- Added support for Solaris, SunOS 4.x, and Linux.
- `HTGap` and `HGGap` commands were limited to an unsigned int.
- `HTLatency` did not set the appropriate trigger.
- All references to Active port were changed to `SmartCard`.

Version 2.11

New Features and Fixes

- Visual Basic function prototypes for `HTGetHubLEDs` and `HGGetLEDs` were incorrect.
- The `SETUP` program would not allow installation from a non-root directory. `A:\SETUP` or `C:\SETUP` would work, `C:\TEMP\SETUP` would not.

Version 2.10

New Advanced Functions

- `ETEnableBackgroundProcessing` that can be used to enhance the responsiveness of applications.

- ETIsBackgroundProcessing determines if a background process is running.
- ETReturnAddress returns a pointer to a Visual Basic data type. An example of this call is shown in the VFD3 code snippet below.
- New Features and Fixes
- HGAddtoGroup now can be used along with HGSetGroup to create groups of ports.
- HTLatency can now be used to measure latency using specific cards.
- HTCRC and HGCRC can be used to generate CRC errors.
- HTAlign and HGAlign can be used to generate alignment errors.
- HTDribble and HGDribble can be used to generate dribble bit errors.
- HTTPortType and HTHubSlotPorts can be used to determine what cards exist in a SmartBits hub.
- HTVFD now supports a static field definition for easy programming of MAC addresses.
- HTGetLEDs and HGGetLEDs now returns LED states.
- HTGetHubLEDs now returns LED states for an entire hub.
- HTSelectTransmit now selects via Hub/Slot/Port ET-1000 transmission.
- HTSelectReceive now selects via Hub/Slot/Port ET-1000 reception and capture.
- ETEnableBackgroundProcessing that can be used to enhance the responsiveness of applications.
- ETIsBackgroundProcessing determines if a background process is running.
- ETReturnAddress returns a pointer to a Visual Basic data type. An example of this call is shown in the VFD3 code snippet below.
- Using ETSetup with ETRECALLSETUP and SetupId of 0 (return to factory defaults), could leave an attached SmartBits hub in an unknown state. Now, all hubs and all cards are reset to the default state when this command is issued. Also, the connection to the ET-1000/SmartBits is maintained across this call. The baud rate in effect before issuing this call is restored before the call returns. There is no need to disconnect and reconnect after this call.
- ETSetBaud now maintains a connection to the ET-1000/SmartBits. There is no longer a need to disconnect and reconnect after using this call.

Initial connection time when using an ETLink command may be minimized by calling ETSetBaud to the baud rate of the device prior to ETLink as below:

```
ETSetBaud(ETBAUD_38400);          //Start searching at 38400
ETLink(ETCOM2);                  //Try to connect to ET1000
//This will search all baud rates, but will set the baud
//rate to 38400 for the first search. If you want to
//guarantee the fastest possible connection after
//connect, use:
ETSetBaud(ETBAUD_38400);          //Start searching at 38400
ETLink(ETCOM2);                  //Try to connect to ET1000
ETSetBaud(ETBAUD_38400);          //Reset to 38400
```

Modified the Visual Basic structure definition HTVFDStructure to:

```
Type HTVFDStructure
Configuration As Integer
Range As Integer
Offset As Integer
iPointer As Long
iLength As Integer
End Type
```

An example Visual Basic snippet to set a VFD3 field is as follows:

```
Static inData(24) As Integer
Static VFD As HTVFDStructure
inData(0) = 255 'Set up "VFD" data structure
inData(1) = 255 'to contain 2 source and dest
inData(2) = 255 'addresses
inData(3) = 255 '
inData(4) = 255 ' Destination:
inData(5) = 255 ' "FF-FF-FF-FF-FF-FF"
inData(6) = 0 ' Source:
inData(7) = 160 ' "00-A0-86-FF-00-00"
inData(8) = 134 '
inData(9) = 255 '
inData(10) = 0 '
inData(11) = 0 '
inData(12) = 0 'Start of 2nd packet structure
inData(13) = 160 ' Destination:
inData(14) = 134 ' "00-A0-86-FF-00-00"
inData(15) = 255 '
inData(16) = 0 '
inData(17) = 0 '
inData(18) = 0 ' Source:
inData(19) = 160 ' "00-A0-86-FF-00-01"
inData(20) = 134 '
inData(21) = 255 '
inData(22) = 0 '
inData(23) = 1 '
VFD.Configuration = HVFD_ENABLED
VFD.Range = 12 'Bytes in VFD
VFD.Offset = 0 'Offset in bits from first bit
VFD.iPointer = ETReturnAddress(inData(0))
'Visual Basic does not support a
'pointer type, so this is a
'work-around.
VFD.iLength = 24 'two different destination/source
'addresses
iRtn = HTVFD(HVFD_3, VFD, 0, 0, 0)
```


Version 2.01

New Features and Fixes

- `ETEnableBackgroundProcessing` that can be used to enhance the responsiveness of applications.
- `ETIsBackgroundProcessing` determines if a background process is running.
- `ETReturnAddress` returns a pointer to a Visual Basic data type. An example of this call is shown in the VFD3 code snippet below.
- `HTSelectReceivePort` and `HGSelectReceivePort` were incompletely documented.

Version 2.0

New Features and Fixes

A new set of Hub “Group” commands have been added. All of these commands are prefixed with an “HG” and are fully described in the Detailed Description section of this manual. The customer should look to utilize these new “HG” commands any time that multiple SmartBits ports are being sent the same “HT” command. Significant performance improvements can be achieved in the ET-1000/SmartBits programming time.

There are two steps to utilizing the new “HG” commands. First, one must set up a “PortIdGroup” string using the new `HGSetGroup(char* PortIdGroup)` command. Then use the “HG” commands similar to how the HT commands are currently used. Every subsequent “HG” command will take effect on all ports listed in the PortIdGroup string.

This has benefits in coding and significant execution time improvements when dealing with more than a few cards at a time. For most programmers, this will enable more inline coding, thus preventing most need to repetitively loop through all the ports to be set up using the HT commands. At run time, the combined overhead of the code loops, operating system, serial communication, and instrument hardware response times are cut by as much as twenty times. This can be quite a significant performance increase if many commands are used to configure and reconfigure your SmartCards during and between various test procedures. There is a new coding example with this distribution that demonstrate the HG commands in C (`PORTGRUP.EXE`).

The library is now available as a Microsoft Windows Version 3.1 DLL. This file is called `ETSMBW16.DLL` and should be copied to the `\WINDOWS\SYSTEM` directory.

The `HTCountStructure` was changed to use unsigned longs for all event counters.

Notes on Using Microsoft Visual Basic

Applications that are created in Visual Basic may call any exported DLL function. Visual Basic calls these functions “external procedures”. These external procedures must be defined by using the “Declare” statement in the Declarations section of a form or module. Netcom distributes a file named “`ETSMBAPI.TXT`” that declares all the functions and structures referenced in this manual. This file may be included in your Visual Basic projects.

Structures are called “User-Defined Data Types” in Visual Basic. All structures referenced in this manual have equivalent Type definitions in `ETSMBAPI.TXT`.

Some of the constants used have changed names. This is because Visual Basic does not allow functions and global constants to have the same names.

C	Visual Basic
HTSTOP	HTRUN_STOP
HTSTEP	HTRUN_STEP
HTRUN	HTRUN_RUN
ETSTOP	ETRUN_STOP
ETSTEP	ETRUN_STEP
ETRUN	ETRUN_RUN

The DLL opens the Comm port to communicate to the ET-1000 & SmartBits Hub. The DLL creates and uses an internal memory block throughout the set of calls used to communicate with the device. Visual Basic does not handle this situation in a normal fashion. Normally, Visual Basic loads and unloads a DLL for each call or procedure used. This would have the effect of removing the memory block in-between DLL calls. So, to handle this situation, programs use the following code fragments:

In a global module:

```
Declare Function LoadLibrary Lib "Kernel" (ByVal lpLibFileName As String) As Integer
Declare Sub FreeLibrary Lib "Kernel" (ByVal hLibModule As Integer)
Global OpenedET As Integer
Global ETLibHandle As Integer
```

In the initial form load:

```
Sub Form_Load ()
    ETLibHandle = LoadLibrary("etsmbw16.dll")
    OpenedET = ETLink(ETCOM2)
End Sub
```

At the unload of this form, use:

```
Sub Form_Unload (Cancel As Integer)
    If (OpenedET > 0) Then
        iRtn = ETUnLink()
        If (iRtn < 0) Then
            MsgBox "Bad Close of ET Connection", 48
        End If
    End If
    FreeLibrary ETLibHandle
End Sub
```

This will load the DLL and keep it in memory throughout the application life.

Visual Basic Demonstration Application

There is a demonstration program, ETVBDEMO.EXE, written in Visual Basic, that demonstrates several different capabilities of the device. The demonstration is distributed the source code. The source code modules used are:

Form	Description
SPLASH.FRM	A introductory “splash” screen shown for a short time while initializing the ET-1000
CONNECT.FRM	A background form, not shown, that controls background processing. This background processing is retrieving the counters for display
MAIN.FRM	The main sample form
ETSETUP.FRM	Set up transmission of the ET-1000 ports
SMBSET UP.FRM	Set up transmission of any of the SmartCards found.
PATTERN.FRM	A dummy pattern editor
GLOBALS.BAS	Global variables used by the forms above.
ETSMBAPI.BAS	A module created by including the ETSMBAPI.TXT file.

The following capabilities are not implemented in this demonstration program:

- VFD fields do not have any effect.
- Hex pattern editors for the Fill and VFD fields are not implemented.
- Triggering is not implemented.
- Error generation is not implemented.
- Echo mode is not implemented.
- The program does not query the device state prior to displaying any information. No checking is done prior to transmission of packet length, gap, data contents, error generation or any other type of packet transmission capability.
- The SmartBits Hub/Port cards when switched, do not update the state of the Run/Stop/Burst buttons

Fixes

HTTrigger was confusing to operate. HT_TRIGGER_ON, HT_TRIGGER_OFF and HT_TRIGGER_INDEPENDENT are now the only mode arguments required

Version 1.32

New Features and Fixes

An HTEcho command has been added to the library. This command is detailed in a new page in the reference section of the manual. Once a card is set up to trigger on an event (e.g. data pattern received), then that card will echo the received packet by transmitting it out the same port of that card.

The **HTVFDStructure** now has a new parameter that is necessary for **VFD_3**. This structure has been amended to add the integer variable member “DataCount” to the end of the structure. The HTVFDStructure.DataCount member should be filled with the byte count (the size) of the Data buffer your program wants VFD_3 to pull bytes from to make up packet transmissions. This is the same buffer that is pointed to by the HTVFDStructure.Data member. The HTVFDStructure.Range member is still the packet size.

The **HTSelectReceivePort(int PortId)** now allows the programmer to turn off the last selected Receive Port by entering a PortId of 0 (zero). This is equivalent to the newly defined value in the ET1000.H file as defined in the following table.

Defined Value	Value Meaning
HTRECEIVE_OFF	OFF

This allows the programmer to turn off the receive mode of the last board routed to Port B of the ET-1000 for analysis.

Software Environment

The ET-1000 library now supports Borland C/C++ 4.02 as well as 4.0 and 3.1. To do this, the name of the Borland 4.0 library has changed. Refer to the table below for the correct library to use with your program. You must decide which library is compatible before attempting to link.

File Name	Development For:
B4ET1000.LIB	Borland C/C++ version 4.02 applications
B40ET1K.LIB	Borland C/C++ version 4.0 applications
B3ET1000.LIB	Borland C/C++ version 3.1 applications
MSET1000.LIB	Microsoft C/C++ (Visual C/C++ version 1.5) applications
ET1000.H	Library header file

Fixes

- The VFDs were not correctly generated.
- The Trigger pattern was not correctly generated.
- The Trigger_Off Mode parameter was not disabling the Trigger.
- The HTSelectReceivePort command was not functional.
- The HTSelectTMTPort command indexed SmartBits ports incorrectly. It now indexes them like Passive Hub cards which assumes two ports per board.

Even though SmartCards have only one port, they are indexed as if there are two ports. This is important to note if you use any of the following three library calls which take a single PortId parameter instead of the “Hub, Slot, Port” addressing of other commands. These three commands are:

- HTSelectReceivePort(PortId, Mode),
- HTSelectTMTPort(PortId, Mode),
- HTSetLED(PortId, Color).

If you have all SmartCards, if PortId is equal to 1 or 2, it will address the first SmartCard in the first Hub. Similarly, PortId equal to 3 or 4 will address the second SmartCard in the first Hub. And so on through to PortId 159 or 160 will address SmartCard 20 in the fourth Hub. For customers whose cards have two ports already, those are Passive cards, so your code should not be affected.

Compatibility with Previous Version

Most code previously linked with version 1.3 of this library will link with version 1.32 without modifications other than what has been noted above. There have also been upgrades to the Firmware that must be loaded before the HTEcho command will work. For best results you should have firmware version 8 or above to avoid problems when trying to control an attached SmartBits. Do **NOT** link your code with version 1.32 unless you have upgraded (or are about to upgrade) the firmware on your ET-1000 to Version 8 or above.

A field upgrade of ET-1000 firmware is available from Netcom Systems. The firmware is upgraded using a MS-DOS executable program (provided by Netcom Systems), and it requires about five to ten minutes to complete the upgrade process.

Version 1.3

New Features and Fixes

- Functions for controlling and monitoring a SmartBits with SmartCards installed have been added. These additional commands allow you to exercise control over any SmartCards installed within the SmartBits Hub Tester. Compatibility with the previous HT-40 functions is maintained.
- Structure HTCCountStructure has been added; it is used to obtain statistical information on the SmartBits SmartCards.
- Structure HTVFDStructure has been added. HTVFDStructure is used to define VFD information required by all SmartCards that are to implement VFD functions.
- Functions for setting and reading the Live Network Mode (LNM) of the ET-1000 have been added. These functions are ETGetLNM() for reading the current status of LNM and ETLNM() for setting LNM in a specific mode.
- The ET-1000 library now supports Borland C/C++ 3.1 as well as 4.0. Separate library files have been released for each type of compiler. If you are using a Borland compiler, you must decide which library is compatible before attempting to link.
- A function for getting the timestamp of a captured packet has been provided. Function ETGetCaptureTime() performs this task. A new structure, "TimeStructure," has been provided with this release for holding the timestamp information.

Documentation

There have been several modifications to this manual due to either A) the addition of functions in the library, or B) correction of errors in the Version 1.2 User's Manual.

Compatibility with Previous Version

All code previously linked with version 1.2 of this library will link with version 1.3 without modification; however, attempting to run this new version on an ET-1000 that does not have firmware version 8 or above may produce problems when trying to control an attached HT-40. Thus, do **NOT** link your code with version 1.3 unless you have upgraded (or are about to upgrade) the firmware on your ET-1000 to Version 8 or above. Field upgrade firmware is available from Netcom Systems. The firmware is upgraded using an MS-DOS executable program (provided by Netcom Systems), and it requires about five to ten minutes to complete the upgrade process.

Appendix D:

Obsolete Functions and Structures

Type	Function / Structure	Description
Capture	int ETGetCaptureTime (TimeStructure* TStruct)	OBSOLETE Not supported
SmartBits	int HGBurst (long IVal)	OBSOLETE Sets the burst count and then sets burst mode. Replaced by the two commands: HGBurstCount and HGTransmitMode.
HT-40	Int HGClear (void)	OBSOLETE Used on an HT-40 with Passive Hub cards only. Clears all ports of a PortIdGroup attached to the ET-1000. Passive cards only. For non-Passive SmartCards this function has been replaced by the command: HTClearPort
SmartBits	Int HGEcho (int iMode)	OBSOLETE When Mode is ON, the select port will echo back the received packet when a trigger condition is met. Replaced by the command: HGTransmitMode
SmartBits	Int HGSelectReceivePort (int PortId)	OBSOLETE Selects a single receive port on the HT-40 Hub Tester(s) which is to be routed to the ET-1000's Port B for analysis. Only one port can be selected at a time. This command can be used on both SmartCards and Passive Hub cards. Replaced by the command HGSelectReceive.
SmartBits	Int HGSelectTMTPort (int Mode)	OBSOLETE Selects the HT-40 Hub Tester(s) to transmit the ET-1000's Port B signals through the PortIds in the PortIdGroup. This command can be used on both SmartCards and Passive Hub Cards. Replaced by the command: HGSelectTransmit.
SmartBits	Int HGSetLED (int Color)	OBSOLETE Illuminates an HT-40's LED associated with a PortIdGroup in the specified color
SmartBits	Int HTBurst (long IVal, int iHub, int iSlot, int iPort)	OBSOLETE Sets the burst count and then sets the transmit mode to a single burst of packets. Replaced by the two commands: HTBurstCount and HTTransmitMode.
HT-40	Int HTClear (int HubId)	OBSOLETE Used on an HT-40 with Passive Hub cards only. Clears one or all HT-40 Hub Testers attached to the ET-1000. Passive cards only. For non-Passive SmartCards this function has been replaced by the command, HTClearPort.
SmartBits	Int HTEcho (int iMode, int iHub, int iSlot, int iPort)	OBSOLETE When Mode is ON, the select port will echo back the received packet when a trigger condition is met. Replaced by the command: HTTransmitMode.
SmartBits	Int HTGroup (int iHub, char* pszGroupString)	OBSOLETE Use HGSetGroup Used to group ports on a SmartBits for purposes of coordinating starting, stopping and stepping the transmission of Ethernet packets from different ports. Replaced by the HGSetGroup and related HG (group) commands.
SmartBits	Int HTGroupStart (int iHub, char* pszGroupString)	OBSOLETE Use HGSetGroup Simultaneously starts the transmission of packets in a group of SmartCards within the specified hub. Replaced by HGSetGroup command and related HG (group) commands.

Type	Function / Structure	Description
SmartBits	Int HTGroupStep (int iHub, char* pszGroupString)	OBSELETE Use HGSetGroup Simultaneously causes the transmission of a single packet in each of a group of SmartCards within the specified hub. Replaced by HGSetGroup and related HG (group) commands.
SmartBits	Int HTGroupStop (int iHub, char* pszGroupString)	OBSELETE Use HGSetGroup Simultaneously halts the transmission of packets in a group of SmartCards within the specified hub. Replaced by HGSetGroup and related HG (group) commands.
SmartCard	Int HTLatencyTest (SetLatencyStructure* pSLS, unsigned long* pulResults, int iMode)	OBSELETE Used to run latency tests on a group of ports of a SmartBits. Replaced by the command: HTLatency.
SmartBits	Int HTSelectReceivePort (int PortId)	OBSELETE Selects a single receive port on the HT-40 Hub Tester(s) which is to be routed to the ET-1000's Port B for analysis. Only one port can be selected at a time. This command can be used on both SmartCards and Passive Hub cards. Replaced by the command: HTSelectReceive.
SmartBits	Int HTSelectTMTPort (int PortId, int Mode)	OBSELETE A transmit port on the HT-40 Hub Tester(s) which is to transmit the ET-1000's Port B signals. This command can be used on both SmartCards and Passive Hub Cards. Replaced by the command: HTSelectTransmit.
SmartBits	Int HTSetLED (int PortId, int Color)	OBSELETE Illuminates an HT-40's LED associated with a particular port in the specified color. This command can be used on both SmartCards and Passive Hub Cards.

SetLatencyStructure

int **Hub**

Identifies the hub on which latency tests are to be run. *Range:* 0 to 3.

int **TransmitSlot**

Identifies the transmit slot within the hub that is to transmit the test pattern. This test pattern is used on receiving slots to determine the latency.

int **ReceiveSlot[20]**

An array of 20 integers. A zero in a particular position of the array indicates that the corresponding slot on the hub is NOT used for latency testing. A one in a particular position of the array indicates that the corresponding slot on the hub IS used for latency testing.

int **Offset**

This is the offset, in bits, from the beginning of the packet (after the preamble bits) that the bit pattern is located. Packets containing the bit pattern are transmitted from the slot identified in TransmitSlot and triggered upon in the slots identified in the ReceiveSlot array.

int **Range**

unsigned char **Pattern[12]**

This is the size of the bit pattern, in bytes. This contains the bit pattern, represented as unsigned characters across the entire array. Pattern[12] contains the most significant byte, Pattern[0] the least significant.

ETGetCaptureTime

Current implementation always forces TIME_TAG_OFF. This command does not return valid information.

Description	Returns time stamp information from the most recently acquired captured packet
Syntax	int ETGetCaptureTime(TimeStructure* Tstruct)
Parameters	<i>TStruct</i> TimeStructure* Points to the structure to be filled with time stamp information.
Return Value	The return value is >= 0 if the function executed successfully. The return value is < 0 if the function failed. <i>See Appendix B for error codes.</i>
Comments	The TimeTag member of the CaptureStructure structure most recently sent to the ET-1000 (via the ETCaptureParams function) must be set to TIME_TAG_ON in order for this function to yield any useful information. In other words, the ET-1000 must be told to save time tag ETCaptureParams function) must be set to TIME_TAG_ON in order for this function to yield any useful information. In other words, the ET-1000 must be told to save time tag information with each captured packet before ETGetCaptureTime can be expected to produce any data. Furthermore, function ETGetCapturePacket must be executed prior to executing this function. ETGetCapturePacket actually acquires the time tag information and puts it into an internal array – ETGetCaptureTime simply copies this information into the provided TimeStructure structure. Thus, the time tag information provided by this function pertains to the packet most recently acquired by ETGetCapturePacket.

HGBurst

Description	Sets up a burst count for transmitting a burst of packets from all ports associated with the PortIdGroup defined by the HGSetGroup(PortIdGroup) command.
Syntax	int HGBurst(long IVal)
Parameters	<i>IVal</i> long Specifies the burst count. <i>Range:</i> 0 to 16,777,215. A value of zero turns off the burst mode, and a non-zero value automatically enables the burst mode.
Return Value	The return value is >= 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This instruction does not cause a burst of packets to be sent. Use HGRun , HGStart , HGStep , HTGroupStart , HTGroupStep , and HTRun to actually start the transmission of the burst.

HGClear

Description	Clears one or all HT-40 Hub Testers attached to the ET-1000. This instruction applies only to HT-40s populated with passive hub cards. For SmartBits with SmartCards, use HTClearPort .
Syntax	int HGClear()
Parameters	None.
Return Value	The return value is >= 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This function assumes that at least one HT-40 Hub Test device is attached to the ET-1000. It will be ignored by the ET-1000 if there is not an HT-40 device present.

HGEcho

Description	Indicates whether to echo back the received packet when a Trigger condition is met from all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command.
Syntax	int HGEcho(int iMode)
Parameters	<p><i>iMode</i> int Indicates whether the selected Port should turn ON or OFF it's echo mode. The OFF mode puts the card into a continuous mode of operation.</p> <p>HTECHO_ON Sets port to Echo mode</p> <p>HTECHO_OFF Sets port to Continuous mode (disabling Echo)</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	None

HGSelectReceivePort

Description	Selects a port on an HT-40 Hub Tester(s) or SmartBits that is to be used for receive data. The receive data from this port is routed directly back to the ET-1000's Port B for detailed analysis.
Syntax	int HGSelectReceivePort(int PortId)
Parameters	<p><i>PortId</i> int Determines the specific port on the HT-40 Hub Tester or SmartBits from which to route data back to the ET-1000's Port B for detailed analysis. Each HT-40 has up to 40 passive ports, or 20 active ports. Up to 4 HT-40s may be cascaded for a total of 160 passive ports, or 80 active ports. PortId ranges from 1 (Port 1 of the first HT-40) to 160, or 80 (last port on the last HT-40). The selected port will be used for analysis of received data. If PortId is 0, the currently selected receive port will be set off. Any values outside this range are invalid and have no effect on the attached ET-1000 or its HT-40 counterpart.</p> <p>NOTE: This command follows the same PortId numbering convention as the HGSetGroup command. The ports are referenced according to their actual presence in the Hub Tester. For example, if the first board in the first Hub is not present, PortId = 1 will refer to the next actual board in the Hub Tester system.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	Because the ET-1000 circuitry only allows one channel to be fully detailed, this command only works on the single port listed in the PortId parameter, but is referenced the same as all ports in the HG commands (See "NOTE" above). This function assumes that at least one SmartBits or HT-40 Hub Test device is attached to the ET-1000. It will be ignored by the ET-1000 if there is not a SmartBits or HT-40 device present.

HGSelectTMTPort

Description	Enables the Port B transmission of the ET-1000 to be transmitted to all ports associated with the PortIdGroup defined by the previous HGSetGroup(PortIdGroup) command. Transmission mode is determined by <i>Mode</i> .
Syntax	int HGSelectTMTPort(int Mode)
Parameters	<p><i>Mode</i> int Determines the function of the Port specified in <i>PortId</i>:</p> <p>HTTRANSMIT_OFF Transmitter is turned off</p> <p>HTTRANSMIT_STD Transmitter transmits standard packets</p> <p>HTTRANSMIT_COL Transmitter transmits collision packets</p> <p>All other values are invalid and have no effect on the attached ET-1000 or its HT-40 counterpart.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See Appendix B for error codes.
Comments	<p>1. This function assumes that at least one SmartBits is attached to the ET-1000. It will be ignored by the ET-1000 if there is not a SmartBits or HT-40 device present.</p> <p>2. Note that when the HTTRANSMIT_COL parameter is set in the Mode argument, the collision type produced by the specified SmartBits or HT-40 port is determined by the most recent parameters placed in the CollisionStructure and sent to the ET-1000 with the ETCollision command. Specifically, only the Offset and Duration fields of the CollisionStructure are used to determine the offset and duration of the collisions produced by the specified HT-40 port. It doesn't matter what the Count or Mode fields of the CollisionStructure are set to—only the Offset and Duration are used by the HT-40. (This is true even if the Mode field of the CollisionStructure is set to COLLISION_OFF—Collisions are turned off for the ET-1000's ports but not necessarily the same is true for the HT-40's ports.)</p>

HGSetLED

Description	Illuminates the HT-40's LED in the specified color for all ports associated to the PortIdGroup defined by the previous HGSetGroup(PortIdGroup).
Syntax	int HGSetLED(int Color)
Parameters	<p><i>Color</i> int Determines the color in which to illuminate the selected Port's LED:</p> <p>HTLED_OFF LED is off</p> <p>HTLED_RED LED is on and red</p> <p>HTLED_GREEN LED is on and green</p> <p>HTLED_ORANGE LED is on and orange</p> <p>Any values outside this range are invalid and have no effect on the attached ET-1000 or its HT-40 counterpart</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See Appendix B for error codes.
Comments	This function assumes that at least one HT-40 is attached to the ET-1000. It will be ignored by the ET-1000 if there is not an HT-40 present.

HTBurst

Description	Sets up a burst count for transmitting a burst of packets from a SmartCard.	
Syntax	int HTBurst(long lVal, int iHub, int iSlot, int iPort)	
Parameters	<i>lVal</i>	long Specifies the burst count. <i>Range:</i> 0 to 16,777,215. A value of zero turns off the burst mode, and a non-zero value automatically enables the burst mode.
	<i>iHub</i>	int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Working with Multiple Hubs</i> in Chapter 1.
	<i>iSlot</i>	int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).
	<i>iPort</i>	int Identifies the SmartCard port. (On the current SmartCards, <i>Port</i> is always 0.)
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	This instruction does not cause a burst of packets to be sent. The HTRun command must be used to start the transmission of the burst.	

HTClear

Description	Clears one or all HT-40 Hub Testers attached to the ET-1000. This instruction applies only to HT-40s populated with passive hub cards. For SmartBits with SmartCards, use HTClearPort .	
Syntax	int HTClear(int iHubId)	
Parameters	<i>iHubId</i>	int Identifies the specific Hub Tester that is to be cleared:
	HTHUBID_1	Hub Tester 1
	HTHUBID_2	Hub Tester 2
	HTHUBID_3	Hub Tester 3
	HTHUBID_4	Hub Tester 4
	HTHUBID_ALL	All attached Hubs
	Any other value is invalid and have no effect on the attached ET-1000 or its HT-40 counterpart.	
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .	
Comments	This function assumes that at least one HT-40 Hub Test device is attached to the ET-1000. It will be ignored by the ET-1000 if there is not an HT-40 device present.	

HTEcho

Description	Indicates to the selected Port whether to echo back a programmed packet when a Trigger condition is met.
Syntax	int HTEcho(int iMode, int iHub, int iSlot, int iPort)
Parameters	<p><i>iMode</i> int Indicates whether the selected Port should turn ON or OFF it's echo mode. The OFF mode puts the card into a continuous mode of operation.</p> <p style="padding-left: 40px;">HTECHO_ON Sets port to Echo mode</p> <p style="padding-left: 40px;">HTECHO_OFF Sets port to Continuous mode (Disabling Echo)</p> <p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0.</p> <p style="padding-left: 40px;">See <i>Working with Multiple Hubs</i> in Chapter 1.</p> <p><i>iSlot</i> int Identifies the slot where the card is located. <i>Range:</i> 0 (first slot in <i>Hub</i>) to 19 (last card in <i>Hub</i>).</p> <p><i>iPort</i> int Identifies the SmartCard port. (On the current SmartCards, <i>Port</i> is always 0.)</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTGroup

Description	Reserves a group of ports within a specified hub. These ports may then be manipulated simultaneously with one another (as a group) using the HTGroupStart() , HTGroupStep() and HTGroupStop() instructions.
Syntax	int HTGroup(int iHub, char* pszGroupString)
Parameters	<p><i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Working with Multiple Hubs</i> in Chapter 1.</p> <p><i>pszGroupString</i> char* A NULL terminated ASCII character string of up to 255 characters which describes the ports that are to be grouped. Port descriptions consist of numbers separated by commas and/or blank spaces. A range of ports may be specified by inserting a hyphen between two port numbers. <i>For example:</i> 0 , , 3,5 11 - 7, 17 19 specifies ports 0, 3, 5, 7, 8, 9, 10, 11, 17 and 19. Note that though the range appears to specify a descending order, it is still interpreted correctly. Ranges are inclusive; thus, the endpoints (7 and 11, in this case) are part of the group. Also, any number of commas or blank spaces may be inserted between the port numbers, as long as the overall length of the string doesn't exceed 255.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	Only one group can exist at any time within a hub. Groups cannot cross hub boundaries. i.e.; you cannot group ports in one hub with ports in another hub. This function can only group SmartCards together. HTGroup() will not return an error indication if you attempt to group ports that are not of the SmartBits type. Groups may be defined and redefined at any time. Each SmartBits hub may have its own group defined.

HTGroupStart

Description	Simultaneously starts the transmission of packets in a group of SmartCards within the specified hub. The group must have been previously defined using the “Set Group” commands.
Syntax	int HTGroupStart(int iHub)
Parameters	<i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Working with Multiple Hubs</i> in Chapter 1.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTGroupStep

Description	Simultaneously causes the transmission of a single packet in each of a group of SmartCards within the specified hub. The group must have been previously defined using the “Set Group” commands.
Syntax	int HTGroupStep(int iHub)
Parameters	<i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Working with Multiple Hubs</i> in Chapter 1.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTGroupStop

Description	Simultaneously halts the transmission of packets in a group of SmartCards within the specified hub. The group must have been previously defined using the “Set Group” commands.
Syntax	int HTGroupStop(int iHub)
Parameters	<i>iHub</i> int Identifies the hub where the card is located. The range is 0 (first hub) through N (number of hubs – 1). Remember to subtract one since the hub identification starts at 0. See <i>Working with Multiple Hubs</i> in Chapter 1.
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See <i>Appendix B for error codes</i> .
Comments	None

HTLatencyTest

Description	<p>Sets up the hub identified in LStruct for latency testing. A single slot is selected for transmission of a packet containing a bit pattern, and several receive slots are set up to trigger on the reception of packets containing the pattern. This function is also used to read the results of a latency test. The results of the latency test are deposited in array "Data," which contains 20 elements corresponding to each of the 20 possible slots.</p>						
Syntax	<pre>int HTLatencyTest(SetLatencyStructure* LStruct, unsigned long* Data, int Mode)</pre>						
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 20%; vertical-align: top;"><i>LStruct</i></td> <td> <p>SetLatencyStructure* Points to a SetLatencyStructure data structure which contains all information necessary to set up a hub for latency testing. This structure also contains the array in which the results of latency test are deposited. See section 5 of this document for a complete description of this structure.</p> </td> </tr> <tr> <td style="vertical-align: top;"><i>Data</i></td> <td> <p>unsigned long* Points to an array large enough to hold 20 unsigned long types. The results of the latency measurement are deposited in this array—each element in the array corresponds to a particular slot. For example, slot 0's results are deposited into Data[0], slot 1 into Data[1], and so on. A value of 0xFFFFFFFF indicates an invalid reading. The results are provided in terms of bit times. (i.e.; 100 ns increments.)</p> </td> </tr> <tr> <td style="vertical-align: top;"><i>Mode</i></td> <td> <p>int Defines the mode of operation for this command. If Mode == HT_RUN_LATENCY, then a latency test is run. i.e.; the transmitting slot is instructed to transmit a packet with a particular bit sequence, and all the requested receivers are instructed to trigger on that same pattern. Results returned in Data may not be valid upon return from this function. If Mode == HT_GET_LATENCY, then the results from a previous function call (in which Mode == HT_RUN_LATENCY) are scooped up from the receiving ports and returned in the Data array. When HTLatencyTest is run in this mode, only the "Hub" element of the HStruct needs to be defined.</p> </td> </tr> </table>	<i>LStruct</i>	<p>SetLatencyStructure* Points to a SetLatencyStructure data structure which contains all information necessary to set up a hub for latency testing. This structure also contains the array in which the results of latency test are deposited. See section 5 of this document for a complete description of this structure.</p>	<i>Data</i>	<p>unsigned long* Points to an array large enough to hold 20 unsigned long types. The results of the latency measurement are deposited in this array—each element in the array corresponds to a particular slot. For example, slot 0's results are deposited into Data[0], slot 1 into Data[1], and so on. A value of 0xFFFFFFFF indicates an invalid reading. The results are provided in terms of bit times. (i.e.; 100 ns increments.)</p>	<i>Mode</i>	<p>int Defines the mode of operation for this command. If Mode == HT_RUN_LATENCY, then a latency test is run. i.e.; the transmitting slot is instructed to transmit a packet with a particular bit sequence, and all the requested receivers are instructed to trigger on that same pattern. Results returned in Data may not be valid upon return from this function. If Mode == HT_GET_LATENCY, then the results from a previous function call (in which Mode == HT_RUN_LATENCY) are scooped up from the receiving ports and returned in the Data array. When HTLatencyTest is run in this mode, only the "Hub" element of the HStruct needs to be defined.</p>
<i>LStruct</i>	<p>SetLatencyStructure* Points to a SetLatencyStructure data structure which contains all information necessary to set up a hub for latency testing. This structure also contains the array in which the results of latency test are deposited. See section 5 of this document for a complete description of this structure.</p>						
<i>Data</i>	<p>unsigned long* Points to an array large enough to hold 20 unsigned long types. The results of the latency measurement are deposited in this array—each element in the array corresponds to a particular slot. For example, slot 0's results are deposited into Data[0], slot 1 into Data[1], and so on. A value of 0xFFFFFFFF indicates an invalid reading. The results are provided in terms of bit times. (i.e.; 100 ns increments.)</p>						
<i>Mode</i>	<p>int Defines the mode of operation for this command. If Mode == HT_RUN_LATENCY, then a latency test is run. i.e.; the transmitting slot is instructed to transmit a packet with a particular bit sequence, and all the requested receivers are instructed to trigger on that same pattern. Results returned in Data may not be valid upon return from this function. If Mode == HT_GET_LATENCY, then the results from a previous function call (in which Mode == HT_RUN_LATENCY) are scooped up from the receiving ports and returned in the Data array. When HTLatencyTest is run in this mode, only the "Hub" element of the HStruct needs to be defined.</p>						
Return Value	<p>The return value is >= 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i></p>						
Comments	<ol style="list-style-type: none"> 1. The calling function must allocate sufficient room (20 unsigned longs) within the space pointed to by Data before calling this routine. 2. When this function is run with Mode == HT_RUN_LATENCY, the results returned in Data most likely will not be valid. In order to get valid results, this function must be run again with Mode == HT_GET_LATENCY. It must follow the initial execution of this function, but only after a period of time at which all trigger packets have arrived on their receive ports. In other words, you must run this function twice: the first time, the function sends out the packets and starts all the necessary timers; the second time, the function gets results from all the timers. Obviously, on the second time that this function is executed, you must be reasonably sure that the trigger packets have had enough time to arrive at the receive ports. 3. It typically requires 3 clock periods (300 nanoseconds) for the latency pattern to circulate out the transmit slot and directly into a receive slot. This must be subtracted off any latency measurements made with these slots. 						

HTSelectReceivePort

Description	Selects a port on an HT-40 or SmartBits that is to be used for receive data. The receive data from this port is routed directly back to the ET-1000's Port B for detailed analysis.
Syntax	int HTSelectReceivePort(int PortId)
Parameters	<p>int Determines the specific port on the HT-40 Hub Tester or SmartBits from which to route data back to the ET-1000's Port B for detailed analysis. Each HT-40 has up to 40 ports, and up to 4 HT-40s may be cascaded for a total of 160 ports. <i>PortId</i> ranges from 1 (Port 1 of the first HT-40) to 160 (Port 40 on the last HT-40). The selected port will be used for analysis of received data. If PortId is 0, the currently selected receive port will be set off. Any values outside this range are invalid and have no effect on the attached ET-1000 or its HT-40 counterpart.</p> <p>NOTE: If you have all SmartCards, then Port numbers 1 and 2 will address your port on the card in slot 1, and Port numbers 3 and 4 will address your port on the card in slot 2, etc.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See Appendix B for error codes.
Comments	This function assumes that at least one SmartBits is attached to the ET-1000. It will be ignored by the ET-1000 if there is not an HT-40 device present.

HTSelectTMTPort

Description	Selects a transmit port on an HT-40 or Smart Bits. Transmission mode is determined by <i>Mode</i> .
Syntax	int HTSelectTMTPort(int PortId, int Mode)
Parameters	<p><i>PortId</i> int Identifies the HT-40 port to which the data length command is to be sent.</p> <p><i>Mode</i> int Determines the function of the Port specified in <i>PortId</i>:</p> <p>HTTRANSMIT_OFF Transmitter is turned off</p> <p>HTTRANSMIT_STD Transmitter transmits standard packets</p> <p>HTTRANSMIT_COL Transmitter transmits collision packets</p> <p>All other values are invalid and have no effect on the attached ET-1000 or its HT-40 counterpart.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. See Appendix B for error codes.
Comments	<p>1. This function assumes that at least one SmartBits is attached to the ET-1000. It will be ignored by the ET-1000 if there is not an HT-40 device present.</p> <p>2. Note that when the HTTRANSMIT_COL parameter is set in the Mode argument, the collision type produced by the specified HT-40 port is determined by the most recent parameters placed in the CollisionStructure and sent to the ET-1000 with the ETCollision command. Specifically, only the Offset and Duration fields of the CollisionStructure are used to determine the offset and duration of the collisions produced by the specified HT-40 port. It doesn't matter what the Count or Mode fields of the CollisionStructure are set to—only the Offset and Duration are used by the HT-40. (This is true even if the Mode field of the CollisionStructure is set to COLLISION_OFF—Collisions are turned off for the ET-1000's ports but not necessarily the same is true for the HT-40's ports.)</p>

HTSetLED

Description	Illuminates an HT-40's LED associated with a particular port in the specified color.
Syntax	int HTSetLED(int PortId, int Color)
Parameters	<p><i>PortId</i> int Identifies the HT-40 port to which the data length command is to be sent..</p> <p><i>Color</i> int Determines the color in which to illuminate the selected Port's LED:</p> <p style="padding-left: 40px;">HTLED_OFF LED is turned off</p> <p style="padding-left: 40px;">HTLED_RED LED is turned ON and is red</p> <p style="padding-left: 40px;">HTLED_GREEN LED is turned ON and is green</p> <p style="padding-left: 40px;">HTLED_ORANGE LED is turned ON and is orange</p> <p style="padding-left: 80px;">Any values outside this range are invalid and have no effect on the attached ET-1000 or its HT-40 counterpart.</p>
Return Value	The return value is ≥ 0 if the function executed successfully. A failure code, which is less than zero, is returned if the function failed. <i>See Appendix B for error codes.</i>
Comments	This function assumes that at least one HT-40 Hub Test device is attached to the ET-1000. It will be ignored by the ET-1000 if there is not an HT-40 device present.

Index

Functions and structures currently supported for SmartBits systems appear in alphabetical order by function name or structure name.

Functions and structures used only with the ET-1000 are listed under “ET-1000-only functions and structures.”

Obsolete functions and structures are listed under “Obsolete functions and structures.”

- Byte alignment switch, 9
- C demo modules, 79
 - ATM, 79
 - common steps, 80
 - files, 81
 - LAN-6100A, 79
 - Layer 2, 79
 - Layer 3, 80
 - steps to run, 81
- C/C++
 - SmartLib interface, 1
- C/C++ development
 - MS Windows, 53
 - UNIX, 61
- CaptureStructure, 202
- Code examples
 - C demo modules, 79
- Coding examples
 - Tcl demo scripts, 64
- CollisionStructure, 120, 204
- Compatibility, 2
- Conventions used in this guide, 3
- CountStructure, 210
- Data structures
 - CaptureStructure, 202
 - Collision, 204
 - CollisionStructure, 120
 - CountStructure, 210
 - EnhancedCountersStructure, 125
 - SwitchStructure, 215
 - TriggerStructure, 214
 - VFDStructure, 222
- Data structures, 109
 - memory allocation, 110
- Data Structures
 - FrameSpec_Type, 189
 - HTCountStructure, 156
 - HTLatencyStructure, 164
 - HTTriggerStructure, 141
 - HTVFDStructure, 142, 186
 - Layer3Address, 164
 - TokenRingAdvancedStructure, 133, 175
 - TokenRingLLCStructure, 135, 178
 - TokenRingMACStructure, 136, 178
 - TokenRingPropertyStructure, 137, 179
 - VGCardPropertyStructure, 138
 - VGCardStructure, 181
- Delphi development
 - MS Windows, 56
- Directory contents
 - MS Windows programming, 52
 - UNIX programming, 60
- Document revision history, 2
- Documentation
 - online, 2
- EnhancedCountersStructure, 125
- Error codes, 224
- ET prefix, 3, 109
- ET-1000-only functions
 - ETAlignCount, 199
 - ETBNC, 200
 - ETBurst, 201

- ETCaptureParams, 201
- ETCaptureRun, 203
- ETCollision, 203
- ETDataLength, 204
- ETDataPattern, 205
- ETDribbleCount, 205
- ETGap, 206
- ETGapScale, 206
- ETGetAlignCount, 207
- ETGetBNC, 207
- ETGetBurstCount, 207
- ETGetBurstMode, 208
- ETGetCapturePacket, 208
- ETGetCapturePacketCount, 208
- ETGetCaptureParams, 209
- ETGetCollision, 209
- ETGetCounters, 209
- ETGetCRCError, 211
- ETGetCurrentLink, 211
- ETGetDataLength, 211
- ETGetDataPattern, 211
- ETGetDribbleCount, 212
- ETGetErrorStatus, 212
- ETGetGap, 212
- ETGetGapScale, 212
- ETGetJET210Mode, 213
- ETGetLNM, 213
- ETGetPreamble, 213
- ETGetReceiveTrigger, 214
- ETGetRun, 214
- ETGetSel, 214
- ETGetSwitch, 215
- ETGetTransmitTrigger, 215
- ETGetVFDRun, 216
- ETLNM, 216
- ETLoopback, 216
- ETMFCounter, 217
- ETPreamble, 217
- ETReceiveTrigger, 218
- ETRemote, 218
- ETReset, 218
- ETReturnAddress, 219
- ETRun, 219
- ETSetJET210Mode, 219
- ETSetSel, 220
- ETSetup, 220
- ETTransmitCRC, 221
- ETTransmitTrigger, 221
- ETVFDPParams, 221
- ETVFDRun, 223
- ETEnableBackgroundProcessing, 111
- ETGetBaud, 111, 139
- ETGetController, 111
- ETGetFirmwareVersion, 112
- ETGetHardwareVersion, 112
- ETGetLibVersion, 112
- ETGetLinkFromIndex, 113
- ETGetLinkStatus, 113
- ETGetSerialNumber, 113
- ETGetTotalLinks, 113
- ETIsBackgroundProcessing, 114
- ETLink, 114
- ETMake2DArray, 114
- ETMake3DArray, 115
- ETSetBaud, 115
- ETSetCurrentLink, 116
- ETSetCurrentSockLink, 116
- ETSetGPSDelay, 117
- ETSetTimeout, 117
- ETSocketLink, 117
- ETSocketLink function, 117
- ETUnLink, 118
- Frame templates
 - NSModifyFrame command, 22
 - using NSCreateFrame, 21
- FrameSpec_Type structure, 189
- Functions
 - ETSocketLink, 117

-
- Header files, 9
 - HG prefix, 3, 109
 - HGAddtoGroup, 118
 - HGAlign, 118
 - HGBurstCount, 119
 - HGBurstGap, 119
 - HGBurstGapAndScale, 119
 - HGClearGroup, 120
 - HGClearPort, 120
 - HGCollision, 120
 - HGCollisionBackoffAggressiveness, 121
 - HGCRC, 121
 - HGDataLength, 122
 - HGDribble, 122
 - HGDuplexMode, 122
 - HGFillPattern, 123
 - HGGap, 123
 - HGGapAndScale, 124
 - HGGetCounters, 124
 - HGGetEnhancedCounters, 124
 - HGGetGroupCount, 127
 - HGGetLEDs, 127
 - HGIsHubSlotPortInGroup, 128
 - HGIsPortInGroup, 128
 - HGMultiBurstCount, 128
 - HGRemoveFromGroup, 129
 - HGRemovePortIdFromGroup, 129
 - HGResetPort, 130
 - HGRun, 130
 - HGSelectTransmit, 131
 - HGSetGroup, 131
 - HGSetGroupType, 132
 - HGSetSpeed, 133
 - HGSetTokenRingAdvancedControl, 133
 - HGSetTokenRingErrors, 135
 - HGSetTokenRingLLC, 135
 - HGSetTokenRingMAC, 136
 - HGSetTokenRingProperty, 137
 - HGSetTokenRingSrcRouteAddr, 138
 - HGSetVGProperty, 138
 - HGStep, 139
 - HGStop, 139
 - HGSymbol, 139
 - HGTransmitMode, 140
 - HGTrigger, 141
 - HGVFD, 142
 - HT prefix, 3, 109
 - HTAlign, 143
 - HTBurstCount, 145
 - HTBurstGap, 145
 - HTBurstGapAndScale, 146
 - HTCardModels, 147
 - HTClearPort, 148
 - HTCollision, 148
 - HTCollisionBackoffAggressiveness, 149
 - HTCountStructure, 156
 - HTCRC, 149
 - HTDataLength, 150
 - HTDribble, 150
 - HTDuplexMode, 151
 - HTFillPattern, 151
 - HTFindMIIAddress, 152
 - HTFrame, 152
 - HTGap, 153
 - HTGapAndScale, 154
 - HTGetCardModel, 155
 - HTGetCounters, 156
 - HTGetEnhancedCounters, 157
 - HTGetEnhancedStatus, 157
 - HTGetHubLEDs, 159
 - HTGetHWVersion, 160
 - HTGetLEDs, 159
 - HTGetStructure, 161
 - HTHubId, 162
 - HTHubSlotPorts, 162
 - HTLatency, 163
 - HTLatencyStructure, 164
 - HTLayer3SetAddress, 164

- HTMultiBurstCount, 165
- HTPortProperty, 166
- HTPortType, 167
- HTReadMII, 168
- HTResetPort, 168
- HTRun, 169
- HTSelectReceive, 169
- HTSelectTransmit, 170
- HTSendCommand, 170
- HTSeparateHubCommands, 171
- HTSetCommand, 172
- HTSetSpeed, 173
- HTSetStructure, 174
- HTSetTokenRingAdvancedControl, 175
- HTSetTokenRingErrors, 177
- HTSetTokenRingLLC, 177
- HTSetTokenRingMAC, 178
- HTSetTokenRingProperty, 179
- HTSetTokenRingSrcRouteAddr, 180
- HTSetVGProperty, 180
- HTSlotOwnership, 181
- HTSlotReserve, 182
- HTSymbol, 183
- HTTransmitMode, 184
- HTTrigger, 185
- HTTriggerStructure, 141
- HTVFD, 186
- HTVFDStructure, 142, 186
- HTWriteMII, 188
- Hub numbering
 - multiple chassis, 17
- Hub/slot/port identification, 13
- Identifying hub/slot/port, 13
- Installation
 - Tcl, 37
 - Windows directory, 38
- IP socket connection to SmartBits, 10
- Layer3Address structure, 164
- Library files, 9
- Library revision history, 2
- LibX, 85, 88
 - card numbering, 88
 - cards supported, 85
 - command format, 88
 - component files, 85
 - loading, 86
 - procedure defaults, 94
 - streams, 103
 - summary of procedures, 107
 - system requirements, 85
 - writing output to a file, 95
- Link timeouts
 - ETSetTimeout command, 11
 - keepalive routine (C), 11
 - keepalive routine (Tcl), 11
 - SmartLib response, 11
- Link timeouts (serial port), 11
- Linking to SmartBits, 10
- Memory allocation
 - for data structures, 110
- Message Functions, 1
- MS Windows programming, 51
 - directory contents, 52
 - installation, 51
- Multi-user access
 - defined, 20
 - functions, 20
- NS prefix, 3, 109
- NSCreateFrame, 188
- NSCreateFrame command, 21
- NSCreateFrameAndPayload, 190
- NSDeleteFrame, 191
- NSGetMaxHubs, 191
- NSGetMaxPorts, 192
- NSGetMaxSlots, 192
- NSGetNumHubs, 192
- NSGetNumPorts, 192
- NSGetNumSlots, 193

-
- NSModifyFrame, 193
 - NSSetPayload, 194
 - NSSetPortMappingMode, 195
 - NSSocketLink, 195
 - NSUnLink, 196
 - Obsolete functions and structures, 249
 - ETGetCaptureTime, 251
 - HGBurst, 251
 - HGClear, 251
 - HGEcho, 252
 - HGSelectReceivePort, 252
 - HGSelectTMTPort, 253
 - HGSetLED, 253
 - HTBurst, 254
 - HTClear, 254
 - HTEcho, 255
 - HTGroup, 255
 - HTGroupStart, 256
 - HTGroupStep, 256
 - HTGroupStop, 256
 - HTLatencyTest, 257
 - HTSelectReceivePort, 258
 - HTSelectTMTPort, 258
 - HTSetLED, 259
 - SetLatencyStructure, 250
 - Original Functions, 1
 - Original Functions, 109
 - Port mapping modes
 - Compatible mode, 13
 - library functions, 19
 - Native mode, 16
 - Prefixes (of function names), 3, 109
 - Programming
 - byte alignment switch, 9
 - C/C++ in MS Windows, 53
 - C/C++ in UNIX, 61
 - Delphi in MS Windows, 56
 - general guidelines, 9
 - header files, 9
 - library files, 9
 - linking to SmartBits, 10
 - MS Windows, general notes, 53
 - Tcl in MS Windows, 55
 - Tcl in UNIX, 61
 - Visual Basic in MS Windows, 56
 - readme.html file, 2
 - Return values
 - error codes, 224
 - Revision history
 - library, 231
 - Serial port
 - link timeouts, 11
 - link to SmartBits, 10
 - SmartAPIs, 1
 - SmartLib
 - applications, 1
 - code examples, 63
 - data structures, 109
 - documentation, 2
 - library interfaces, 1
 - Original Functions, 109
 - programming in MS Windows, 51
 - programming in UNIX, 59
 - system requirements, 9
 - Tcl setup, 38
 - test functionality, 1
 - SwitchStructure, 215
 - Synchronizing chassis or stacks, 17
 - Tcl
 - how to use with SmartLib, 37
 - installing
 - Windows directory, 38
 - installing, 37
 - setting up SmartLib, 38
 - SmartLib interface, 1
 - Tcl shell, 40
 - Tcl demo scripts, 64
 - all cards, 64

- ATM, 65
- ET-1000, 67
- Ethernet, 68
- Fast Ethernet (FastCard), 68
- Gigabit (GIG), 69
- Layer3, 70
- LibX, 72
- POS, 72
- SmartAPI, 73
- Token Ring, 73
- Tcl development
 - MS Windows, 55
 - UNIX, 61
- technical support
 - how to contact, 4
- TokenRingAdvancedStructure, 133, 175
- TokenRingLLCStructure, 135, 178
- TokenRingMACStructure, 136, 178
- TokenRingPropertyStructure, 137, 179
- TriggerStructure, 214
- Troubleshooting, 4
- UNIX programming, 59
 - directory contents, 60
 - installation, 59
 - versions tested, 59
- VFDStructure, 222
- VGCardPropertyStructure, 138
- VGCardStructure, 181
- Visual Basic
 - SmartLib interface, 1
- Visual Basic development
 - MS Windows, 56