SmartBits
Advanced Multiport Performance Tester/Simulator/Analyzer

SmartLib
Message Functions
Programming Library Version 3.05

FEBRUARY 1999

Covering:
HTSetStructure
HTGetStructure
HTSetCommand

Supporting these SmartCards:
ATM
Frame Relay
Ethernet
Fast Ethernet
SmartMetrics Ethernet (L3 and ML)
Gigabit Ethernet
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 THE 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: Introduction

Message Functions vs. Original Functions .................................................. 12
Understanding Prefixes: HT, HG, and ET ................................................... 13
Understanding Streams ............................................................................... 14
ATM Streams / Connections........................................................................ 16
VFD Support ................................................................................................ 17
How to: Work with Ethernet SmartMetrics Streams .................................... 17
Histogram Results ....................................................................................... 20
Setting up Histograms Step-by-Step ........................................................... 22

## Chapter 2: Function Concepts

Message Function Overview ....................................................................... 24
The Action (iType1) and the Structure......................................................... 24
Syntax of the Message Functions ............................................................... 26
How Each Parameter is Used ..................................................................... 27
Language-Specific Usage ........................................................................... 28

## Chapter 3: ATM

ATM - HTSetStructure Summary................................................................. 32
ATM - HTGetStructure Summary ............................................................... 33
ATM - HTSetCommand Summary ............................................................... 33
ATM - HTSetStructure ................................................................................. 35

- **ATM_CALL_ADDR_LIST** ................................................................. 35
- **ATMCallAddrList** ................................................................................. 35
- **ATM_CALL_SETUP** ................................................................................. 35
- **ATMCallSetupParams** ........................................................................... 35
- **ATM_CLASSICAL_IP** ................................................................................. 36
- **ATMClassicalIP** ................................................................................ 36
- **ATM_CONN_COPY** ................................................................................. 37
- **ATMConnectionCopyParams** ............................................................... 37
- **ATM_CONN_MODIFY** ................................................................................. 38
- **ATMConnectionModify** ........................................................................ 38
- **ATM_CONN_MODIFY_ARRAY** .............................................................. 39
- **ATMConnectionModifyArray** .............................................................. 39
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM_CONN_TRIGGER_PARAMS</td>
<td>39</td>
</tr>
<tr>
<td>ATMConnTriggerParams</td>
<td>39</td>
</tr>
<tr>
<td>ATM_CONNECTION</td>
<td>40</td>
</tr>
<tr>
<td>ATMConnection</td>
<td>40</td>
</tr>
<tr>
<td>ATM_DS1_E1_LINE_PARAM</td>
<td>42</td>
</tr>
<tr>
<td>ATMDS1E1LineParams</td>
<td>42</td>
</tr>
<tr>
<td>ATM_DS3_E3_LINE_PARAM</td>
<td>43</td>
</tr>
<tr>
<td>ATMDS3E3LineParams</td>
<td>43</td>
</tr>
<tr>
<td>ATM_ELAN_DEREGISTER</td>
<td>44</td>
</tr>
<tr>
<td>ATMELANDeregister</td>
<td>44</td>
</tr>
<tr>
<td>ATM_ELAN_REGISTER</td>
<td>44</td>
</tr>
<tr>
<td>ATMELANRegister</td>
<td>44</td>
</tr>
<tr>
<td>ATM_FRAME_COPY</td>
<td>45</td>
</tr>
<tr>
<td>ATMFrameCopyReq</td>
<td>45</td>
</tr>
<tr>
<td>ATM_FRAME_DEF</td>
<td>46</td>
</tr>
<tr>
<td>ATMFrameDefinition</td>
<td>46</td>
</tr>
<tr>
<td>ATM_GLOBAL_TRIGGER_PARAMS</td>
<td>47</td>
</tr>
<tr>
<td>ATMGlobalTrigger</td>
<td>47</td>
</tr>
<tr>
<td>ATM_ILMI</td>
<td>47</td>
</tr>
<tr>
<td>ATMILMIParams</td>
<td>47</td>
</tr>
<tr>
<td>ATM_ILMI_STATIC_REGISTER</td>
<td>48</td>
</tr>
<tr>
<td>ATMILMIStaticParams</td>
<td>48</td>
</tr>
<tr>
<td>ATM_INCOMING_SVC_METHOD</td>
<td>48</td>
</tr>
<tr>
<td>ATMIncomingSvcMethod</td>
<td>48</td>
</tr>
<tr>
<td>ATM_LINE</td>
<td>49</td>
</tr>
<tr>
<td>ATMLineParams</td>
<td>49</td>
</tr>
<tr>
<td>ATM_PER_CONN_BURST</td>
<td>50</td>
</tr>
<tr>
<td>ATMPerConnBurstCount</td>
<td>50</td>
</tr>
<tr>
<td>ATM_PER_PORT_BURST</td>
<td>50</td>
</tr>
<tr>
<td>ATMPerPortBurstCount</td>
<td>50</td>
</tr>
<tr>
<td>ATM_SCHED_PARAMS</td>
<td>51</td>
</tr>
<tr>
<td>ATMschedParams</td>
<td>51</td>
</tr>
<tr>
<td>ATM_SCCOP</td>
<td>51</td>
</tr>
<tr>
<td>ATMSCCOPParams</td>
<td>51</td>
</tr>
<tr>
<td>ATM_STREAM</td>
<td>52</td>
</tr>
<tr>
<td>ATMStream</td>
<td>52</td>
</tr>
<tr>
<td>ATM_STREAM_CONTROL</td>
<td>54</td>
</tr>
<tr>
<td>ATMStreamControl</td>
<td>54</td>
</tr>
<tr>
<td>ATM_STREAM_PARAMS_COPY</td>
<td>54</td>
</tr>
<tr>
<td>ATMStreamParamsCopy</td>
<td>54</td>
</tr>
<tr>
<td>ATM_STREAM_PARAMS_MODIFY</td>
<td>55</td>
</tr>
<tr>
<td>ATMStreamParamsModify</td>
<td>55</td>
</tr>
<tr>
<td>ATM_STREAM_PARAMS_FILL</td>
<td>55</td>
</tr>
<tr>
<td>ATMStreamParamsFill</td>
<td>55</td>
</tr>
<tr>
<td>ATM_TRIGGER</td>
<td>56</td>
</tr>
<tr>
<td>ATMTrigger</td>
<td>56</td>
</tr>
<tr>
<td>ATM_UNI</td>
<td>57</td>
</tr>
<tr>
<td>ATMUNIParams</td>
<td>57</td>
</tr>
<tr>
<td>ATM - HTGetStructure</td>
<td>57</td>
</tr>
<tr>
<td>ATM_AAL5_INFO</td>
<td>57</td>
</tr>
<tr>
<td>ATMAAL5LayerInfo</td>
<td>57</td>
</tr>
<tr>
<td>ATM_CARD_CAPABILITY</td>
<td>58</td>
</tr>
<tr>
<td>ATMCardCapabilities</td>
<td>58</td>
</tr>
<tr>
<td>ATM_CARD_INFO</td>
<td>59</td>
</tr>
<tr>
<td>ATMCardInfo</td>
<td>59</td>
</tr>
<tr>
<td>ATM_CARD_TYPE</td>
<td>60</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ATM Card Type</td>
<td>60</td>
</tr>
<tr>
<td>ATM_CLASSICAL_IP_INFO</td>
<td>60</td>
</tr>
<tr>
<td>ATM Classical IP Info</td>
<td>60</td>
</tr>
<tr>
<td>ATM_CONN_64_INFO</td>
<td>61</td>
</tr>
<tr>
<td>ATM Connection64Info</td>
<td>61</td>
</tr>
<tr>
<td>ATM_CONN_64_INFO_SUMMARY</td>
<td>61</td>
</tr>
<tr>
<td>ATM Connection64InfoSummary</td>
<td>61</td>
</tr>
<tr>
<td>ATM_CONN_INFO</td>
<td>62</td>
</tr>
<tr>
<td>ATM ConnectionInfo</td>
<td>62</td>
</tr>
<tr>
<td>ATM_CONN_INFO_SUMMARY</td>
<td>62</td>
</tr>
<tr>
<td>ATM ConnectionInfoSummary</td>
<td>62</td>
</tr>
<tr>
<td>ATM_CONN_TRIGGER_INFO</td>
<td>63</td>
</tr>
<tr>
<td>ATM ConnTriggerInfo</td>
<td>63</td>
</tr>
<tr>
<td>ATM_DS1_E1_LINE_INFO</td>
<td>63</td>
</tr>
<tr>
<td>ATMDS1E1LineInfo</td>
<td>63</td>
</tr>
<tr>
<td>ATM_DS3_E3_LINE_INFO</td>
<td>64</td>
</tr>
<tr>
<td>ATMDS3E3LineInfo</td>
<td>64</td>
</tr>
<tr>
<td>ATM_ELAN_INFO</td>
<td>65</td>
</tr>
<tr>
<td>ATM ELANInfo</td>
<td>65</td>
</tr>
<tr>
<td>ATM_ILMI_INFO</td>
<td>65</td>
</tr>
<tr>
<td>ATM ILMIInfo</td>
<td>65</td>
</tr>
<tr>
<td>ATM_LAYER_INFO</td>
<td>66</td>
</tr>
<tr>
<td>ATM LayerInfo</td>
<td>66</td>
</tr>
<tr>
<td>ATM_SAAL_INFO</td>
<td>66</td>
</tr>
<tr>
<td>ATMSAALInfo</td>
<td>66</td>
</tr>
<tr>
<td>ATM_SIG_EMUL_INFO</td>
<td>67</td>
</tr>
<tr>
<td>ATMSigEmulatorInfo</td>
<td>67</td>
</tr>
<tr>
<td>ATM_Sonet_INFO</td>
<td>67</td>
</tr>
<tr>
<td>ATMSonetInfo</td>
<td>67</td>
</tr>
<tr>
<td>ATM_STREAM_DETAIL_INFO</td>
<td>68</td>
</tr>
<tr>
<td>ATMSStreamDetailedInfo</td>
<td>68</td>
</tr>
<tr>
<td>ATM_STREAM_SEARCH_INFO</td>
<td>68</td>
</tr>
<tr>
<td>ATMSStreamSearchInfo</td>
<td>68</td>
</tr>
<tr>
<td>ATM_TRIGGER_INFO</td>
<td>69</td>
</tr>
<tr>
<td>ATMTiggerInfo</td>
<td>69</td>
</tr>
<tr>
<td>ATM_VCC_INFO</td>
<td>69</td>
</tr>
<tr>
<td>ATM VCCInfo</td>
<td>69</td>
</tr>
<tr>
<td>ATM_VCDB_LIST_INFO</td>
<td>70</td>
</tr>
<tr>
<td>ATMVCDiBInfo</td>
<td>70</td>
</tr>
<tr>
<td>ATM - HTSetCommand</td>
<td>70</td>
</tr>
<tr>
<td>ATM_CLIP_ESTABLISH_CLIENT</td>
<td>70</td>
</tr>
<tr>
<td>ATM_CLIP_RELEASE_CLIENT</td>
<td>70</td>
</tr>
<tr>
<td>ATM_CONN_PARAMS_COMPLETE</td>
<td>70</td>
</tr>
<tr>
<td>ATM_CONN_PARAMS_RESET</td>
<td>71</td>
</tr>
<tr>
<td>ATM_FRAME_CLEAR</td>
<td>71</td>
</tr>
<tr>
<td>ATM_ILMI_DEREGISTER</td>
<td>71</td>
</tr>
<tr>
<td>ATM_ILMI_REGISTER</td>
<td>71</td>
</tr>
<tr>
<td>ATM_SAAL_ESTABLISH</td>
<td>71</td>
</tr>
<tr>
<td>ATM_SAAL_RELEASE</td>
<td>71</td>
</tr>
<tr>
<td>ATM_SIG_EMUL_RESET</td>
<td>72</td>
</tr>
<tr>
<td>ATM_START_SETUP</td>
<td>72</td>
</tr>
<tr>
<td>ATM StartCardSetupParams</td>
<td>72</td>
</tr>
<tr>
<td>ATM_STOP_SETUP</td>
<td>73</td>
</tr>
<tr>
<td>ATM StopCardSetupParams</td>
<td>73</td>
</tr>
</tbody>
</table>
Chapter 4: 10/100 MB Ethernet

ETH - HTSetStructure Summary .................................................................74
ETH - HTGetStructure Summary .................................................................74
ETH - HTSetCommand Summary .................................................................75

ETH - HTSetStructure .................................................................................76
    ETH_COLLISION ..................................................................................76
    ETHFillPattern ..................................................................................76
    ETH_LATENCY .................................................................77
    ETHTransmission .............................................................................77
    ETH_TRIGGER ..............................................................................79
    ETHTrigger ....................................................................................79
    ETH_WRITE_MII ............................................................................80

ETH - HTGetStructure .................................................................................81
    ETH_CARD_INFO ................................................................................81
    ETHCounterInfo ..............................................................................81
    ETH_COUNTER_INFO .......................................................................82
    ETHCounterInfo ..............................................................................82
    ETH_ENHANCED_COUNTER_INFO .........................................................82
    ETHEnhancedCounterInfo ................................................................82
    ETH_ENHANCED_STATUS_INFO ............................................................83
    ETHEnhancedStatusInfo ....................................................................83
    ETH_FIND_MII_ADDR_INFO ...............................................................83
    ETHLatency .......................................................................................83
    ETH_LATENCY_INFO .........................................................................84
    ETHLatency .......................................................................................84
    ETH_READ_MII_INFO .........................................................................84

ETH - HTSetCommand .................................................................................84
    ETH_CLEAR_PORT .............................................................................84
    ETH_RESET_PORT ............................................................................84
    ETH_SELECT_RECEIVE ......................................................................85
    ETH_SELECT_TRANSMIT ....................................................................85

Chapter 5: 100 MB Fast Ethernet

FST - HTSetStructure Summary .................................................................86
FST - HTGetStructure Summary .................................................................86
FST - HTSetStructure .................................................................................87
    FST_ALTERNATE_TX ........................................................................87
    FSTAlternateTx ..............................................................................87
    FST_CAPTURE_PARAMS ....................................................................87
    FSTCaptureParams ..........................................................................87
    FST_CONTROL_AUX ..........................................................................88
    FSTControlAux ...............................................................................88

SmartLib - Message Functions
Chapter 6: Gigabit Ethernet

Chapter 7: L3 - Ethernet with SmartMetrics
L3 - HTGetStructure Summary ..................................................................104
L3 - HTSetCommand Summary ....................................................................104
L3 - HTSetStructure ....................................................................................105

L3_DEFINE_IP_STREAM..............................................................................105
StreamIP 105

L3_DEFINE_IPX_STREAM ...........................................................................107
StreamIPX 107

L3_DEFINE_MULTI_IP_STREAM....................................................................109
StreamIP 109

L3_DEFINE_MULTI_IPX_STREAM...................................................................111
StreamIPX 111

L3_DEFINE_MULTI_SMARTBITS_STREAM......................................................113
StreamSmartBits 113

L3_DEFINE_MULTI_UDP_STREAM.................................................................115
StreamUDP 115

L3_DEFINE_SMARTBITS_STREAM.................................................................117
StreamSmartBits 117

L3_DEFINE_UDP_STREAM...........................................................................119
StreamUDP 119

L3_MOD_IP_STREAM....................................................................................121
StreamIP 121

L3_MOD_IPX_STREAM..................................................................................123
StreamIPX 123

L3_MOD_SMARTBITS_STREAM.....................................................................125
StreamSmartBits 125

L3_MOD_STREAMS_ARRAY...........................................................................127
Layer3ModifyStreamArray 127

L3_MOD_STREAMS_DELTA............................................................................129
Layer3ModifyStreamDelta 129

L3_MOD_UDP_STREAM................................................................................131
StreamUDP 131

L3 - HTGetStructure ..................................................................................133

L3_ARP_TIMES_INFO..................................................................................133
Layer3ArpTimesInfo 133

L3_CAPTURE_COUNT_INFO........................................................................133
Layer3CaptureCountInfo 133

L3_CAPTURE_PACKET_DATA_INFO...............................................................134
Layer3CaptureData 134

L3_DEFINED_STREAM_COUNT_INFO............................................................134
Layer3DefinedStreamCountInfo 134

L3_HIST_ACTIVE_TEST_INFO......................................................................135
Layer3HistActiveTest 135

L3_HIST_LATENCY_DISTRIBUTION_INFO....................................................136
Layer3HistLatencyDistributionInfo 136

L3_HIST_RAW_TAGS_INFO..........................................................................137
Layer3HistRawTagsInfo 137

L3_HIST_SEQUENCE_INFO..........................................................................138
Layer3HistSequenceInfo 138

L3_HIST_V2_LATENCY_INFO......................................................................139
Layer3HistLongLatencyInfo 139

L3_HIST_V2_LATENCY_PER_STREAM_INFO................................................140
Layer3HistLongLatencyPerStreamInfo 140

L3_STREAM_INFO.......................................................................................142
StreamSmartBits 142

L3_TX_ADDRESS_INFO.................................................................................144
Chapter 8: Frame Relay 149

FR - HTSetStructure Summary ....................................................149
FR - HTGetStructure Summary .....................................................150
FR - HTSetCommand Summary ......................................................150
FR - HTSetStructure .................................................................152
   FR_CARD_CFG ..........................................................152
      FRCardCfg .........................................................152
   FR_DEFINE_IP_STREAM ..................................................153
      StreamIP ..........................................................153
   FR_DEFINE_SMARTBITS_STREAM .......................................155
      StreamSmartBits ...............................................155
   FR_DEFINE_UDP_STREAM ...............................................157
      StreamUDP ......................................................157
   FR_DUP_PVC ...........................................................159
      FRPvcTableEntry ..............................................159
   FR_DUP_STREAM .......................................................160
      StreamSmartBits ..............................................160
   FR_FILL_PATTERN .......................................................162
   FR_HIST_LATENCY_DISTRIBUTION ....................................162
   FR_HIST_SEQUENCE ......................................................162
   FR_HIST_V2_LATENCY ....................................................162
   FR_HIST_V2_LATENCY_PER_STREAM ....................................163
   FR_IP_SUBNET_DEREG ...................................................163
      FRIPSubnetDeRegister .......................................163
   FR_IP_SUBNET_REG ......................................................164
      FRIPSubnetRegister ..........................................164
   FR_LINE .................................................................165
      FRLineCfg .......................................................165
   FR_LMI .................................................................166
      FRLmiCfg .......................................................166
   FR_MOD_UDP_STREAM .....................................................166
      StreamUDP ......................................................166
<table>
<thead>
<tr>
<th>FR_PVC</th>
<th>168</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_PvcTableEntry</td>
<td>168</td>
</tr>
<tr>
<td>FR_PVC_CTRL</td>
<td>169</td>
</tr>
<tr>
<td>FR_PvcControl</td>
<td>169</td>
</tr>
<tr>
<td>FR_PVC_STREAM_MAP_CFG</td>
<td>170</td>
</tr>
<tr>
<td>FR_PvcStrmMapCfg</td>
<td>170</td>
</tr>
<tr>
<td>FR_STRM_CTRL</td>
<td>171</td>
</tr>
<tr>
<td>FR_StreamControl</td>
<td>171</td>
</tr>
<tr>
<td>FR_T1E1_LINE</td>
<td>171</td>
</tr>
<tr>
<td>FR_T1E1LineCfg</td>
<td>171</td>
</tr>
<tr>
<td>FR_TRIGGER</td>
<td>172</td>
</tr>
<tr>
<td>FR_TriggerCfg</td>
<td>172</td>
</tr>
<tr>
<td>FR_AGGR_LATENCY_DISTRIBUTION_INFO</td>
<td>172</td>
</tr>
<tr>
<td>FR_AGGR_SEQUENCE_INFO</td>
<td>172</td>
</tr>
<tr>
<td>FR_AGGR_V2_LATENCY_INFO</td>
<td>173</td>
</tr>
<tr>
<td>FR_AGGR_V2_LATENCY_PER_STREAM_INFO</td>
<td>173</td>
</tr>
<tr>
<td>FR_CARD_VERSION_INFO</td>
<td>173</td>
</tr>
<tr>
<td>FR_VersionInfo</td>
<td>173</td>
</tr>
<tr>
<td>FR_DEFINED_STREAM_COUNT_INFO</td>
<td>174</td>
</tr>
<tr>
<td>FR_HIST_ACTIVE_TEST_INFO</td>
<td>174</td>
</tr>
<tr>
<td>FR_HIST_LATENCY_DISTRIBUTION_INFO</td>
<td>174</td>
</tr>
<tr>
<td>FR_HIST_SEQUENCE_INFO</td>
<td>174</td>
</tr>
<tr>
<td>FR_HIST_TYPE_INFO</td>
<td>175</td>
</tr>
<tr>
<td>FR_HIST_V2_LATENCY_INFO</td>
<td>175</td>
</tr>
<tr>
<td>FR_HIST_V2_LATENCY_PER_STREAM_INFO</td>
<td>175</td>
</tr>
<tr>
<td>FR_IP_STREAM_INFO</td>
<td>175</td>
</tr>
<tr>
<td>StreamIP</td>
<td>175</td>
</tr>
<tr>
<td>FR_LINK_INFO</td>
<td>177</td>
</tr>
<tr>
<td>FR_LinkInfo</td>
<td>177</td>
</tr>
<tr>
<td>FR_LINK_STATUS_INFO</td>
<td>179</td>
</tr>
<tr>
<td>FR_LinkStatusInfo</td>
<td>179</td>
</tr>
<tr>
<td>FR_LMI_INFO</td>
<td>179</td>
</tr>
<tr>
<td>FR_LmiInfo</td>
<td>179</td>
</tr>
<tr>
<td>FR_PVC_INFO</td>
<td>180</td>
</tr>
<tr>
<td>FR_PvcMainInfo</td>
<td>180</td>
</tr>
<tr>
<td>FR_PVC_STATUS_INFO</td>
<td>181</td>
</tr>
<tr>
<td>FR_PvcStatusInfo</td>
<td>181</td>
</tr>
<tr>
<td>FR_SMARTBITS_STREAM_INFO</td>
<td>182</td>
</tr>
<tr>
<td>StreamSmartBits</td>
<td>182</td>
</tr>
<tr>
<td>FR_T1E1_LINE_INFO</td>
<td>184</td>
</tr>
<tr>
<td>FR_T1E1LineInfo</td>
<td>184</td>
</tr>
<tr>
<td>FR_COMMIT_CMD</td>
<td>184</td>
</tr>
<tr>
<td>FR_COMMIT_CFG</td>
<td>184</td>
</tr>
<tr>
<td>FR_DISABLE_PORT</td>
<td>185</td>
</tr>
<tr>
<td>FR_ENABLE_PORT</td>
<td>185</td>
</tr>
<tr>
<td>FR_GROUP_MEMBER_CMD</td>
<td>185</td>
</tr>
<tr>
<td>FR_GROUP_START_CMD</td>
<td>185</td>
</tr>
<tr>
<td>FR_GROUP_STEP_CMD</td>
<td>185</td>
</tr>
<tr>
<td>FR_GROUP_STOP_CMD</td>
<td>186</td>
</tr>
<tr>
<td>FR_NON_GROUP_CMD</td>
<td>186</td>
</tr>
</tbody>
</table>
Appendix A: Obsolete and Removed Commands 188

Implemented in Later Release ......................................................... 188
Obsolete ....................................................................................... 188
Not Supported ............................................................................ 188
Chapter 1: Introduction

This manual documents the Message Functions. They are the newer method of developing with the SmartLib programming library. These functions are used to communicate between the PC and the SmartBits chassis. The card/controller command is contained in the first parameter of the Message Function.

There are a total of three Message Functions, and each one is much like the other, for a more consistent and predictable interface. The Message Functions support many of the Netcom Systems' SmartCards. The applicable cards are listed at the beginning of each card-specific chapter. Also listed, are any parameters that work only with certain cards out of the card family.

A second type of SmartLib functions are the Original functions discussed in the SmartLib User Guide.

NOTE: Use this manual in addition to the general information provided in SmartLib User Guide (included in this Software Developer's Kit).

A third type of code contained in SmartLib are the test modules (the SmartLib APIs that interface with the hardware functions (Original/Message Functions). These modules are documented in separate manuals.

**Message Functions vs. Original Functions**

HTSetStructure, HTGetStructure, and HTSetCommand are designed to take advantage of the speed of the new SmartCard on-board processing. This style of programming also provides a more flexible and expandable architecture with a consistent interface.

The table below shows you whether you can use Original functions, the Message functions, or both. See appropriate sections for specific exceptions about compatibility. Notice that all Ethernet SmartCards can use the original function group when they are in Traditional (single stream definition) mode. This allows you to swap cards in the chassis without changing test code.

**SmartCard to Function - Table**

<table>
<thead>
<tr>
<th>SmartCards</th>
<th>Original Functions</th>
<th>HTSetStructure, HTGetStructure HTSetCommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 2 10/100 Mbps Ethernet</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token Ring</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Gigabit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Layer 3 Ethernet</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Multi-Layer (ML-7710)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ATM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame Relay</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
NOTE: Certain parameters and structures work with all cards within a family (e.g., ETH, ATM, FR, etc). Some parameters only work with certain cards. At the beginning of each SmartCard chapter is a description of which parameters go with which card(s).

Important Exceptions
There are three important exceptions to the above table.

- All upgrade features on the SX-7410 are accessed by the Message functions. These added features include:
  - Alternate Transmit Stream.
  - Frame Capture Capability.
  - Software Flow Control.
  - Preamble Length Definition.
  - VLAN Tags.

- There is a group of original functions that work with all SmartCards. These functions which are documented in the User Guide are:
  - HTGetEnhancedStatus get SmartCard status.
  - HTClearPort clear all counters on the card.
  - HTRun set card to iMode (run, step, stop).
  - HGStart start transmitting from a group of cards.
  - HGStop stop transmitting from a group of cards.
  - HGStep send one frame from a group of cards.

- There are two functions which currently work on all SmartCards except for ATM and Frame Relay.
  - HTResetPort( iMode, h, s, p ), and
  - HGResetPort( iMode )

These will be updated to work with ATM and Frame Relay in the near future.

Understanding Prefixes: HT, HG, and ET
The three functions in this manual are prefaced with "HT" because they interact on a single card basis.

In SmartLib, function names are prefixed by either HT, HG, or ET. The HT prefix indicates communication to a single SmartCard, while the HG prefix indicates communication to a Group of SmartCards. The ET functions interact with an SMB or ET-1000 controller.
Understanding Streams

A Stream of network traffic is a series of frames transmitted from a source to a destination. To create a stream of traffic:

- A basic frame blueprint is defined.
- The frame is transmitted multiple times so that a stream of traffic is generated.
- For Ethernet and FrameRelay, the frame may be modified so that each frame in the Stream is different.
- With ATM, a Stream is a combination of Stream Structure containing connection parameters, and the Frame structure containing the payload contents.

At this time there are two ways to configure test frames for the generation of test traffic: there is the Traditional Mode and the SmartMetrics Mode. Some SmartCards such as the SX-7210, SX-7410, and TR-8405 support the Traditional Mode exclusively. Cards such as the ML-7710 and L3-6710 support both the SmartMetrics and the Traditional mode. Cards such as the WN-3410 and the AT-9015 support the SmartMetrics mode exclusively.

**Traditional Traffic**

Traditional Mode refers to one method of generating test traffic. In Traditional mode there is only one frame blueprint available per card. This means that in order to simulate more than one stream per card, you must use incrementing or changing patterns within certain fields of the frame blueprint.

For example:

<table>
<thead>
<tr>
<th>Destination MAC</th>
<th>Source MAC (Incrementing VFD1)</th>
<th>Type</th>
<th>Payload (Defined Background Pattern)</th>
<th>CRC</th>
</tr>
</thead>
</table>

**Traditional Ethernet frame blueprint.**

In this example, the VFD1 increments the Source address to create frames seemingly coming from different devices.

Some of the characteristics of Traditional mode are:

- Traffic is based on modifications of a single blueprint.
- By using VFD1, VFD2, and VFD3 (Variable Field Definitions) in addition to the background (fill) pattern, a high degree of complexity and control can be accomplished.
- Traffic can be used to test Layer2 and Layer3 devices.
- Because there is only one frame blueprint per port, information is tracked on a per-port basis.
- There is a CRC check on the entire frame.
**SmartMetrics Traffic**

SmartMetrics Mode refers to a second method of generating test traffic. When a card is in SmartMetrics mode, it supports many unique frame blueprints.

NOTE: Since each frame blueprint is used to generate a different stream of test traffic, we call these blueprints "Streams."

Some of the characteristics of SmartMetrics mode are:
- Unique streams of traffic, generated from multiple frame blueprints.
- Information tracking on a per-stream basis (as opposed to a per-port basis).
- A CRC check on the entire frame.

Additional Ethernet and FrameRelay SmartMetrics features are:
- Imbedded Signature fields with information about each frame.
- In-depth latency and sequence information.
- An IP checksum for IP streams.
- Traffic can be used to test Layer2 and Layer3 and more.

Below is a diagram with an example stream configuration for a single Ethernet SmartCard. Note the multiple frame blueprints, different protocols, and varied frame sizes.

For Ethernet Only, set the stream at index 0 to Inactive; reserving it as a placeholder for Traditional mode.

<table>
<thead>
<tr>
<th>Index 0</th>
<th>MAC Dest</th>
<th>MAC Src</th>
<th>UDP</th>
<th>Prot Header</th>
<th>Payload</th>
<th>Signature</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index 1</td>
<td>MAC Dest</td>
<td>MAC Src</td>
<td>IPX</td>
<td>Prot Header</td>
<td>Payload</td>
<td>Signature</td>
<td>CRC</td>
</tr>
<tr>
<td>Index 3</td>
<td>MAC Dest</td>
<td>MAC Src</td>
<td>IP</td>
<td>Prot Header</td>
<td>Payload</td>
<td>Signature</td>
<td>CRC</td>
</tr>
<tr>
<td>Index 4</td>
<td>MAC Dest</td>
<td>MAC Src</td>
<td>IP</td>
<td>Prot Header</td>
<td>Payload</td>
<td>Signature</td>
<td>CRC</td>
</tr>
<tr>
<td>Index 5</td>
<td>MAC Dest</td>
<td>MAC Src</td>
<td>SMB</td>
<td>Payload</td>
<td></td>
<td>Signature</td>
<td>CRC</td>
</tr>
</tbody>
</table>

**Six "Streams" (frame blueprints) on a single SmartMetrics card.**

If enabled, the Signature field overwrites 18 bytes of data at the end of the payload. It contains information such as the time stamp, Stream ID, and frame sequence.

Ethernet and FrameRelay cards in SmartMetrics mode support the use of a Signature field. This field contains information about the specific frame. The information in the Signature field is a powerful feature used by the receiving card to analyze network traffic (Histograms).
**ATM Streams / Connections**

ATM SmartCards also create multiple streams (multiple frame blueprints) per port. When you create an ATM stream using the `ATM_STREAM` command, you define the connection parameters as well as the type of encapsulation, cell rate, and so forth. The payload of the frames is defined by `ATM_FRAME_DEF`.

Once a stream is defined it has an Stream Index number, based the value of `uiIndex` within the stream structure. This is a value that you control. You can use it to query the SmartCard for information about the Stream structure.

If you want to retrieve information about the connection of a given stream (such as trigger counts), you must use the Connection Index associated with the stream. The Connection Index is not defined in the library. It is a resource allocated by the SmartCard. Because Connection Indexes are assigned by the card as they become available, you must query the card to know which index is associated with a stream. Use `ATM_STREAM_DETAIL_INFO` to retrieve Connection Indexes.

**Example**

Below is a snippet (TCL) of code used to retrieve a block of Connection Indexes. A Connection Index is assigned each time a Connection is established. For a valid Connection Index, the card must be queried each time the Stream is connected.

```tcl
#############################################################
struct_new vcc_info ATMVCCInfo
struct_new stream_info ATMStreamDetailedInfo
# Use ATM_STREAM_DETAIL to get the connection index
LIBCMD HTGetStructure $ATM_STREAM_DETAIL_INFO 0 $NUM_STREAMS 0
stream_info 0 $iHub $iSlot $iPort
# Use ATM_VCC_INFO to get the stats
for {set j 0} {$j < $NUM_STREAMS} { incr j } {
    puts "Checking status on Tx Card (Connection Index $stream_info(status.$j.uiConnIndex))"
    LIBCMD HTGetStructure $ATM_VCC_INFO
    $stream_info(status.$j.uiConnIndex) 1 0 vcc_info 0 $iHub $iSlot $iPort
    puts "Stats for stream $j..."
    # Cell header is decimal by default - force to hex...
    puts "Cell Header [format "%08X"
    $vcc_info(status.0.ulCellHeader)]"
    puts "Tx Frame count ==> $vcc_info(status.0.ulTxFrame)"
    puts ""
}
#############################################################```
ATM SmartCards support independent streams/connections with a wide range of powerful controls. Signature fields and the resultant data analysis via Histograms are not supported (at this time).

**VFD Support**

VFDs are supported by FrameRelay cards, and Ethernet cards in Traditional mode. A VFD (Variable Field Definition), is a field that can be manipulated (incremented, decremented, used in chunks, etc.) VFDs are laid over existing information in the field such as the background pattern.

- **Ethernet Only:**
  In SmartMetrics mode, VFDs are not supported at this time. There is one exception, and that is in the SmartBits customizable stream. In the SmartBits customizable stream, VFD3 is used in limited fashion to enter the custom protocol header and payload.

- **WAN Only:**
  The FrameRelay cards support VFD1 and VFD2 in the SmartMetrics streams. They support VFD3 in limited fashion where Range is the total number of bytes usable from the VFD3 buffer.

**How to:**

**Work with Ethernet SmartMetrics Streams**

This section covers configuring and manipulating SmartMetrics streams for the ML, and L3 SmartCards.

At power-up, the card contains one default stream at index 0. The stream at index 0 is a place-holder for the Traditional mode.

*Configuring a Stream, or List of Streams:*

**Step 1 -** Define the background fill pattern (optional).

- Use **HTFillPattern** or **HGFillPattern** (described in the SmartLib User Guide) to define the fill pattern. The default fill pattern is all zeros.

The fill pattern, (or background pattern), is laid into the frame first. All other fields overwrite the fill pattern at various offsets. You can use the fill pattern to put data into the frame in addition to the structure elements that get copied in.

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

A fill pattern of all "A"s.

<table>
<thead>
<tr>
<th>MAC Dest</th>
<th>MAC Src</th>
<th>Type IP</th>
<th>Protocol Header</th>
<th>Payload</th>
<th>Signature 18 bytes</th>
<th>CRC</th>
</tr>
</thead>
</table>

Same frame with other fields laid over the fill pattern.

**Step 2 -** Declare the appropriate structure (or array of structures).

The structure will depend on which **L3_DEFINE_n_STREAM** you select.

For example:
StreamIP MyIPStream;
memset (&MyIPStream, 0, sizeof(MyIPStream));

Enter correct values for the structure elements - (not shown).

**Step 3 - Create a stream or a list of streams.**
Use L3_DEFINE_n_STREAM.

For example:
HTSetStructure(L3_DEFINE_IP_STREAM, 0,0,0, &MyIPStream, sizeof(StreamIP), iHub,iSlot,iPort);
  
  - HTSetStructure
    Sends the command to the SmartCard.
  
  - L3_DEFINE_IP_STREAM
    Is the command (iType1) that tells the card what to do with the structure.
  
  - StreamIP
    Is the structure (or array of structures) needed to create IP frames. The card uses your structure definition for the frame blueprint.
  
  - If you want to retrieve Histogram results, enable the Signature field by setting ucTagField to 1.
  
  - sizeof(StreamIP)
    Is the size of the structure (* number of elements if an array is used).

**NOTE:** L3_DEFINE_n_STREAM defines a complete list of streams, and will clear all previously defined streams. It also sets the stream at index 0 to inactive, and starts defining streams at index 1.

Some stream elements of note are:
- ucTagField - Must be enabled for Signature fields to be inserted into each test frame. (Signature field data is used to collect Histogram results.)
- VFD3 in the Smartbits Stream is the byte pattern for the customizable frame content.
- ucActive - Sets a stream to active or inactive. Provides a way to skip a stream while leaving the stream index numbers intact.

**Using One Stream to Create Many**
Use L3_DEFINE_MULTI_n_STREAMS.
DEFINE_MULTI creates new streams based on an existing stream. The stream index (iType2) is the prototype stream. The count (iType 3) indicates the number of streams that will be created and appended to the existing list.
The values in the Stream structure are delta values. A value of 0 maintains the current value of a structure member. A number indicates the amount to increment the value in each additional stream. If the current element value is 1, and a value of 3 is used, the new values would be 4,7,10,13, etc.
NOTE: Do not set index to 0, since this will duplicate the inactive reserved stream.

Modifying a Stream
Use L3_MOD_n_STREAM to change the features of an existing stream at a specified index. L3_MOD_n_STREAM replaces the entire stream. The structure values here are not deltas. To change one or more attributes, a complete structure is sent to the card. The index value must indicate an existing stream. If the index value does not reference an existing stream, the command is not acted on.

NOTE: Although L3_MOD_n_STREAM can modify the stream at index 0, be sure to modify streams starting at index 1.

Clearing All Streams (Traditional / SmartMetrics mode)
To clear streams, use L3_DEFINE_n_STREAM, and set the structure pointer (pData) to NULL. This will clear all streams except stream 0, which will be set to inactive. To set the card to Traditional mode (previously called the L2 mode), clear all streams (stream 0 will remain). To switch the card to SmartMetrics mode, create one or more streams starting at index 1.

Sending Traffic
Once the streams are configured, you must start and stop the configured test traffic. To start and stop test traffic, refer to this group of commands in the User Guide:
- HGRun / HTRun
- HGStart
- HGStop
- HGStep

NOTE: Cards with RIP, PING, or ARP enabled will automatically send these frames at specified increments. See HTLayer3SetAddress in the User Guide.

Some related commands are: HGSetGroup, HTSeparateHubCommands, and HTLayer3SetAddress (for the optional card address, not the stream address).

Frame Order
Regardless of whether you send a constant stream of traffic, a burst of traffic, or one frame of traffic from each stream, frames are sent in the same order. The first frame from each stream is sent, then the second frame from each stream is sent. The order is from the smallest stream index to the largest, looping back to the smallest. Inactive streams are skipped.
**Additional Commands for Stream Manipulation**

<table>
<thead>
<tr>
<th>To do this:</th>
<th>Use this command (iType1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify a single field in a group of streams. This is faster than sending</td>
<td>L3_MOD_STREAMS_ARRAY</td>
</tr>
<tr>
<td>an entire group of structures to the card.</td>
<td></td>
</tr>
<tr>
<td>Increment a single field in a group of streams. This is faster than</td>
<td>L3_MOD_STREAMS_DELTA</td>
</tr>
<tr>
<td>sending an entire group of structures to the card.</td>
<td></td>
</tr>
<tr>
<td>Check how many streams are configured on the SmartCard. The returned</td>
<td>L3_DEFINED_STREAM_COUNT_INFO</td>
</tr>
<tr>
<td>value includes the placeholder stream at index 0.</td>
<td></td>
</tr>
<tr>
<td>Retrieve the stream structure at a specific index.</td>
<td>L3_STREAM_INFO</td>
</tr>
<tr>
<td>Setup Address and Gateway information for the card, and specify PING,</td>
<td>HTLayer3SetAddress</td>
</tr>
<tr>
<td>SNMP, or RIP frame transmission. (See the User Guide.)</td>
<td></td>
</tr>
<tr>
<td>NOTE: This command sets up the SmartCard not the test streams. The IP</td>
<td></td>
</tr>
<tr>
<td>address must be unique.</td>
<td></td>
</tr>
</tbody>
</table>

**Histogram Results**

Histograms offer a powerful way to look at test results. These SmartMetrics result gatherers distill and analyze information about networks, based on incoming test traffic. Histograms are now implemented on three of Netcom System's newer SmartCards: ML-7710, L3-6705, and the L3-6710.

As opposed to Counters, Histograms do not simply supply the cumulative numbers of events. Histograms provide relational information and answer questions such as:
- How many frames were out of order, and for which stream?
- At what points during the test did certain events take place?
- Which streams had Latency issues and how often did they occur?

A single histogram can be enabled on a SmartCard for a given test run. The histogram analyzes the results of the current traffic. At this time, there are five histograms to select from.

**The Five Histograms**

To obtain SmartMetrics results, you must first enable the desired Histogram on the SmartCard. There are five possible Histograms to select from:

1. **V2_LATENCY (Latency Over Time)** - 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 would answer this example question: "What was the over-all average latency in the first second?"
This Histogram has a configuration setting. Set the Histogram to analyze data at specified intervals. For example, the Latency measurements during the first nanosecond, during the second, during the third, and so forth.

2. **V2_LATENCY_PER_STREAM (The combination histogram)** - gives you an over-all latency picture per stream. It provides the Minimum, Maximum, and Average latency values over the course of the test (Latency Per Stream). It racks reoccurrence of specific Latency values on a per-stream basis (Latency Distribution). It reports whether frames were received in sequence or not, on a per-stream basis (Sequence Tracking).

3. **LATENCY_DISTRIBUTION (Latency Distribution)** - tracks reoccurrence of specific Latency values on a per-stream basis. This Histogram will let you know, for example, if common latency values are from one to two microseconds and/or the distribution of multiple latency values.

You can set the ranges of latency values. The latency values can be consecutive as in .4,.5,.6 microseconds, or they can vary as in .2,.5,1.9.

4. **SEQUENCE (Sequence Tracking)** - reports whether frames were received in sequence or not, on a per-stream basis. It also identifies duplicate and dropped frames.

The Sequence Tracking algorithm is designed to match an actual TCP stack, and is as follows:

- As long as frames are received in sequence, the in sequenced value 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 counter.
- 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 Sequenced 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).

5. **RAW_TAGS (Bulk Data)** - is a list of statistics on a per-frame basis. It is
different from other Histogram results in that the data is not analyzed. 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.

**Setting up Histograms Step-by-Step**

Follow these four steps to setup and retrieve SmartMetrics histogram results.

**Step 1 -** (Transmit Card)

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

When creating streams with:

\[
\text{HTSetStructure} \quad \text{L3\_DEFINE\_n\_STREAM} \\
\text{L3\_DEFINE\_MULTI\_n\_STREAM} \\
\text{L3\_MOD\_n\_STREAM}
\]

Enable the Signature field by setting \( \text{ucTagField} = 1 \).

**Step 2 -** (Receive Card)

Activate the desired Histogram on the card using:

\[
\text{HTSetCommand} \quad \text{with} \quad \text{L3\_HIST\_n}
\]

The histogram is now in receive mode until you query it for information or records.

**Step 3 -** (Receive Card) - Optional

You can clear all previous records by using the HTSetCommand with

\[
\text{L3\_HIST\_START}
\]

**Step 4 -** (Transmit Card)

Start transmitting test traffic using a function such as \text{HTRun} or \text{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.

**Step 5 -** (Receive Card) Optional

You can find out how many records are on the card by using

\[
\text{HTGetStructure} \quad \text{with} \quad \text{L3\_HIST\_ACTIVE\_TEST\_INFO}
\]

This information is useful to know if you may not want to retrieve all records.
Step 6 - (Receive Card)

Once test traffic has been sent, retrieve Histogram results using:
HTGetStructure with L3_HIST_n_INFO

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:
1. Get Histogram records using: HTGetStructure with either L3_HIST_n_INFO or L3_AGGR_n_INFO.
2. Get information about the Histogram using: L3_ACTIVE_TEST_INFO.

Pointer: Remember to allow adequate time after you start the transmit, before you attempt to retrieve Histogram results.
Chapter 2: Function Concepts

This chapter discusses the three message functions: HTSetStructure, HTGetStructure, and HTSetCommand. It also takes a close look at two common parameters of this group, \texttt{iType1} and \texttt{pData} (the related structure).

Message Function Overview

The purpose of the three messaging functions, HTSetStructure, HTGetStructure, and HTSetCommand, is simply to pass information between the PC and the SmartBits chassis. The actual action to be executed on the SmartCard is indicated by the selected \texttt{iType1} parameter.

The distinguishing features of the three commands are:

- \texttt{HTSetStructure} transmits information to the SmartCard which sets values (with the use of a structure or buffer).
- \texttt{HTGetStructure} gets information from the SmartCard (with the use of a structure or buffer).
- \texttt{HTSetCommand} transmits information to the SmartCard, usually to affect traffic from the SmartCard port.

Occasionally HTSetCommand may be used to affect values on the SmartCard such as reset the card, or clear counters. Some HTSetCommand functions also have a related data structure, although this is not common.

To encapsulate: although the messaging functions have important differences, their basic function is simply as a messenger, transmitting values between the PC and the SmartBits chassis.

Similarities

Although the parameters determine what action takes place on the SmartCard, the syntax of the message functions remains constant. The syntax of the three functions is also very similar. When you understand the syntax of these three functions, you can use this knowledge to set values, get values, and send commands with all of the newer SmartCards:

- ATM
- Frame Relay
- Gigabit Ethernet
- Layer 3 Ethernet/Fast Ethernet
- Multi-Layer

The Action (\texttt{iType1}) and the Structure

The first parameter in all three of the messenger functions is \texttt{iType1}. This parameter determines the action that will occur on the SmartCard.

Most HTSetStructure and HTGetStructure \texttt{iType1} parameters use a specific SmartLib structure type. Occasionally, an HTSetCommand \texttt{iType1} will also use a structure. Often, the name of the \texttt{iType1} is similar to its associated structure.
Note that an iType1 is a constant, and appears in all uppercase letters. As opposed to its related structure which is mixed case.

A given structure may be used by a number of different iType1. Each iType1 has only one valid structure that it can use, however, the related structure may contain embedded (nested) structures. If an element of a structure is another structure, this manual will display the embedded structure below the related structure definition.
Syntax of the Message Functions

The syntax of the message functions is discussed in detail in this section. Any difference between the functions is noted.

The message functions transmit information between the PC and the SmartCards. This information is used to: configure the SmartCard, get information from the SmartCard, or affect the I/O of the SmartCard port.

(For language-specific usage see the examples in the Usage section below.)

Syntax:

```c
int HTSetStructure(int iType1,
                   int iType2, int iType3, int iType4,
                   void* pData,
                   int iLen,
                   int iHub, int iSlot, int iPort);

int HTGetStructure(int iType1,
                     int iType2, int iType3, int iType4,
                     void* pData,
                     int iLen,
                     int iHub, int iSlot, int iPort);

int HTSetCommand(int iType1,
                  int iType2, int iType3, int iType4,
                  void* pData,
                  int iHub, int iSlot, int iPort);

**Note: The iLen parameter is not used by HTSetCommand.**
```
**How Each Parameter is Used**

The three Message Functions have almost identical parameters. Each parameter has a consistent, specific use. Often, some parameters are not used. In this case the value is set to Zero.

Below is a table discussing how each parameter is used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iType1</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>This parameter defines what action takes place on the SmartCard. For example, whether settings are modified or copied, whether transmission starts or stops, etc. Many iType1 parameters have an associated data type. For these iType1s, the appropriate data type must be used for the pData parameter. (See pData below.)</td>
</tr>
</tbody>
</table>

| iType2    | int   |
|           | When "index" is indicated for iType2, pData is a pointer to an array of structures, instead of a single structure. In this case, the iType2 variable indexes a location within an array on the SmartCard. If there is no array of structures, the value of iType2 is set to 0 to indicate the array index is not applicable. |

| iType3    | int   |
|           | This parameter indicates the number of consecutive array elements to modify, define, or affect. It is often used in conjunction with iType2, where iType3 indicates the number of elements to affect after the index pointer. If there is no array of structures, iType3 is set to 0. This indicates that the element count is not applicable. |

| iType4    | int   |
|           | This parameter is not used at this time. Set this parameter to 0 to indicated that iType4 is not applicable. |

| pData     | void* |
|           | This parameter is a pointer to a structure, an array of structures, or a buffer. The related data type is dictated by the iType1. The message functions use a pointer of type void to accommodate different data types. |

| iLen      | int   |
|           | This parameter is the amount of memory to allocate for the data indicated by pData. **Note This parameter is not used by HTSetCommand.** |

| iHub      | int   |
|           | This parameter identifies the destination hub where the SmartCard is located. The range is from 0 (first hub) to 15 (sixteenth hub). |

| iSlot     | int   |
|           | This parameter identifies the slot where the SmartCard is located. The range is from 0 (first slot in the hub) to 19 (last slot in the hub). |

| iPort     | int   |
|           | This parameter identifies the SmartCard port. (On current SmartCards, Port is always set to 0.) |

**Return Value:**
The return value is $\geq 0$ if the function executed successfully. The return value is $< 0$ if the function failed. Failure codes are defined in Appendix A.
Comments:
A given structure can be used by more than one iType1, but each iType1 is associated with a single, specific structure.

Language-Specific Usage
This section contains usage examples for the three message functions. The examples are divided into groups according to programming language.

Note: The lines in the usage examples below are split up for readability only. To understand each element in a function call, look back to “Syntax of the Message Function“ on page 26.

Usage for C/C++

```
HTSetStructure(ATM_CALL_SETUP, 0, 0, 0, (void*) pATMCallSetupParams, sizeof(ATMCallSetupParams), iHub, iSlot, iPort);
```

```
HTGetStructure(ATM_CONN_INFO, iIndex, iCount, 0, (void*) pATMConnectionInfo, sizeof(ATMConnectionInfo), iHub, iSlot, iPort);
```

```
HTSetCommand(ATM_ILMI_REGISTER, 0, 0, 0, NULL, iHub, iSlot, iPort);
```

Usage for Tcl

```
HTSetStructure $ATM_CALL_SETUP \ 0 0 0 \ pATMCallSetupParams \ 0 \ $iHub $iSlot $iPort
```

```
HTGetStructure $ATM_CONN_INFO \ $iIndex $iCount 0 \ pATMConnectionInfo \ 0 \ $iHub $iSlot $iPort
```

```
HTSetCommand $ATM_ILMI_REGISTER \ 0 0 0 \ *** \ $iHub $iSlot $iPort
```
Usage for Visual Basic 3

```vbnet
Dim lAddr as Long
lAddr = ETReturnAddress(pATMCallSetupParams)
HTSetStructure ATM_CALL_SETUP,
    0, 0, 0,
    lAddr,
    LenB(pATMCallSetupParams),
    iHub, iSlot, iPort

Dim lAddr as Long
lAddr = ETReturnAddress(pATMConnectionInfo)
HTGetStructure ATM_CONN_INFO,
    nIndex, iCount, 0,
    lAddress,
    LenB(pATMConnectionInfo),
    iHub, iSlot, iPort

HTSetCommand ATM_ILMI_REGISTER,
    0, 0, 0,
    0,
    iHub, iSlot, iPort
```

Usage for Visual Basic 5

```vbnet
HTSetStructure ATM_CALL_SETUP,
    0, 0, 0,
    pATMCallSetupParams,
    LenB(pATMCallSetupParams),
    iHub, iSlot, iPort

HTGetStructure ATM_CONN_INFO,
    nIndex, iCount, 0,
    pATMConnectionInfo,
    LenB(pATMConnectionInfo),
    iHub, iSlot, iPort

HTSetCommand ATM_ILMI_REGISTER,
    0, 0, 0,
    NULL
    iHub, iSlot, iPort
```

Usage for Delphi

```delphi
HTSetStructure(ATM_CALL_SETUP,
    0, 0, 0,
    @pATMCallSetupParams,
    SizeOf(ATMCallSetupParams),
    iHub, iSlot, iPort);

HTGetStructure(ATM_CONN_INFO,
    nIndex, iCount, 0,
    @pATMConnectionInfo,
    SizeOf(ATMConnectionInfo),
    iHub, iSlot, iPort);

HTSetCommand(ATM_ILMI_REGISTER,
    0, 0, 0,
    @NULL,
    iHub, iSlot, iPort);
```
Chapter 3: ATM

This section covers the Message Functions as related to ATM SmartCards. Chapter three covers all ATM related parameters and structures.

Note: Some structures contain embedded or nested structures. In these cases, the embedded structures are included directly below the related structure.

The ATM SmartCards are split into two groups:
- **ATM1**: AT-9015, AT-9020, AT-9025, AT-9034, AT-9045, AT-9155, AT-9155B
- **ATM2**: AT-9155C, AT-9622

These ATM commands (iType1s) and related structures work with all ATM SmartCards with these exceptions:

<table>
<thead>
<tr>
<th>iType1</th>
<th>Works with these cards only.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM_CONN_TRIGGER_INFO</td>
<td>ATM2 Types</td>
</tr>
<tr>
<td>ATM_CONN_TRIGGER_PARAMS</td>
<td>ATM2 Types</td>
</tr>
<tr>
<td>ATM_DS1_E1_LINE_INFO</td>
<td>AT-9015, AT-9020</td>
</tr>
<tr>
<td>ATM_DS1_E1_LINE_PARAM</td>
<td>AT-9015, AT-9020</td>
</tr>
<tr>
<td>ATM_DS3_E3_LINE_INFO</td>
<td>AT-9034, AT-9045</td>
</tr>
<tr>
<td>ATM_DS3_E3_LINE_PARAM</td>
<td>AT-9034, AT-9045</td>
</tr>
<tr>
<td>ATM_LINE</td>
<td>AT-9155, AT-9155b, AT-9155C, AT-9622, AT-9025</td>
</tr>
<tr>
<td>ATM_SONET_INFO</td>
<td>AT-9155, AT-9155b, AT-9155C, AT-9622, AT-9025</td>
</tr>
<tr>
<td>ATM_TRIGGER</td>
<td>ATM1 Types</td>
</tr>
<tr>
<td>ATM_TRIGGER_INFO</td>
<td>ATM1 Types</td>
</tr>
</tbody>
</table>
## ATM - HTSetStructure Summary

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>iTYPE2</th>
<th>iTYPE3</th>
<th>iTYPE4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM_CALL_ADDR_LIST</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCALLAddrList</td>
<td>Set the called addr. for sigtest</td>
</tr>
<tr>
<td>ATM_CALL_SETUP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCALLSetupParams</td>
<td>Set the traffic desc. for sigtest</td>
</tr>
<tr>
<td>ATM_CLASSICAL_IP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCClassicalIP</td>
<td>Configure CLIP ARP server info</td>
</tr>
<tr>
<td>ATM_CONN_COPY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCConnectionCopyParams</td>
<td>Copy connections for sigtest</td>
</tr>
<tr>
<td>ATM_CONN_MODIFY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCConnectionModify</td>
<td>Fill field of conn. for sigtest</td>
</tr>
<tr>
<td>ATM_CONN_MODIFY_ARRAY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCConnectionModifyArray</td>
<td>Mod field of conn. for sigtest</td>
</tr>
<tr>
<td>ATM_CONN_TRIGGER_PARAMS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCConnTriggerParams</td>
<td>Define per conn trigger event</td>
</tr>
<tr>
<td>ATM_CONNECTION</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCConnection</td>
<td>Define a connection for sigtest</td>
</tr>
<tr>
<td>ATM_DS1_E1_LINE_PARAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMDS1E1LineParams</td>
<td>Configure ATM and PHY for DS1 or E1</td>
</tr>
<tr>
<td>ATM_DS3_E3_LINE_PARAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMDS3E3LineParams</td>
<td>Configure ATM and PHY for DS3 or E3</td>
</tr>
<tr>
<td>ATM_ELANTEREGISTER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMELANRegister</td>
<td>Define and register and LEC</td>
</tr>
<tr>
<td>ATM_FRAME_COPY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMFramCopyReq</td>
<td>Copy Frame params to a given # of streams</td>
</tr>
<tr>
<td>ATM_FRAME_DEF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMFramDefiniton</td>
<td>Configure a new or existing frame</td>
</tr>
<tr>
<td>ATM_GLOBAL_TRIGGER_PARAMS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMGGlobalTrigger</td>
<td>Define global trigger event</td>
</tr>
<tr>
<td>ATM_ILMI</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMLILIMParams</td>
<td>Set ILMI timers and ESI</td>
</tr>
<tr>
<td>ATM_ILMI_STATIC_REGISTER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMLilimStaticParams</td>
<td>Force an ATM address statically</td>
</tr>
<tr>
<td>ATM_INCOMING_SVC_METHOD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMIIncomingSVCMethod</td>
<td>Incoming SVC method</td>
</tr>
<tr>
<td>ATM_LINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMLineParams</td>
<td>Physical and ATM layer config</td>
</tr>
<tr>
<td>ATM_PER_CONN_BURST</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMPerConnBurstCount</td>
<td>Burst count per connection</td>
</tr>
<tr>
<td>ATM_PER_PORT_BURST</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMPerPortBurstCount</td>
<td>Burst count per port</td>
</tr>
<tr>
<td>ATM_SCHED_PARAMS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSchedParams</td>
<td>Cell-scheduling method for transmission</td>
</tr>
<tr>
<td>ATM_SCCOP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSCCPParams</td>
<td>Set SCCOP timers and config</td>
</tr>
<tr>
<td>ATM_STREAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSStream</td>
<td>Configure a new or existing stream</td>
</tr>
<tr>
<td>ATM_STREAM_CONTROL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSStreamControl</td>
<td>Start, stop, reset, etc a stream</td>
</tr>
<tr>
<td>ATM_STREAM_PARAMS_COPY</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMSStreamParamsCopy</td>
<td>Copy Stream params to a given # of streams</td>
</tr>
<tr>
<td>ATM_STREAM_PARAMS_COPY</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMSStreamParamsModify</td>
<td>Configure triggers in ATM-1</td>
</tr>
<tr>
<td>ATM_TRIGGER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMTrigger</td>
<td>Configure triggers in ATM-1</td>
</tr>
<tr>
<td>ATM_UNI</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMUUniparams</td>
<td>Set UNI timers and version</td>
</tr>
</tbody>
</table>
### ATM - HTGetStructure Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM_AAL5_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMAL5LayerInfo</td>
<td>Get the AAL5 layer counts</td>
</tr>
<tr>
<td>ATM_CARD_CAPABILITY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCapabilities</td>
<td>Get card-specific limits</td>
</tr>
<tr>
<td>ATM_CARD_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCardInfo</td>
<td>Query card for firmware version</td>
</tr>
<tr>
<td>ATM_CARD_TYPE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCardType</td>
<td>Get card rate (model number)</td>
</tr>
<tr>
<td>ATM_CLASSICAL_IP_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMCClassicalIPInfo</td>
<td>CLIP counts and ARP status</td>
</tr>
<tr>
<td>ATM_CONN_64_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMConnection64Info</td>
<td>Get 64-bits status of one or many conn</td>
</tr>
<tr>
<td>ATM_CONN_64_INFO_SUMMARY</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMConnection64InfoSumm</td>
<td>Get 64-bits summary results of sigtest</td>
</tr>
<tr>
<td>ATM_CONN_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMConnectionInfo</td>
<td>Get status of one or many conn</td>
</tr>
<tr>
<td>ATM_CONN_INFO_SUMMARY</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMConnectionInfoSummary</td>
<td>Get summary results of sigtest</td>
</tr>
<tr>
<td>ATM_CONN_TRIGGER_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMConnTriggerInfo</td>
<td>Get per conn trigger counts</td>
</tr>
<tr>
<td>ATM_DS1_E1_LINE_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMDS1E1LineInfo</td>
<td>Get the DS1/E1 alarms and counts</td>
</tr>
<tr>
<td>ATM_DS3_E3_LINE_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMDS3E3LineInfo</td>
<td>Get the DS3/E3 alarms and counts</td>
</tr>
<tr>
<td>ATM_ELAN_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>ATMELANInfo</td>
<td>Get the ELAN status</td>
</tr>
<tr>
<td>ATM_ILMI_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMLIMIInfo</td>
<td>Get the ILMI status and counts</td>
</tr>
<tr>
<td>ATM_LAYER_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMLayerInfo</td>
<td>Get the ATM layer counts</td>
</tr>
<tr>
<td>ATM_SAAL_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSAALInfo</td>
<td>Get the SAAL status</td>
</tr>
<tr>
<td>ATM_SIG_EMUL_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSigEmulatorInfo</td>
<td>Get the signaling emulator stats</td>
</tr>
<tr>
<td>ATM_SONET_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSonetInfo</td>
<td>Get the SONET alarms and counts</td>
</tr>
<tr>
<td>ATM_STREAM_DETAIL_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMStreamDetailedInfo</td>
<td>Get status of one or many streams</td>
</tr>
<tr>
<td>ATM_STREAM_SEARCH_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMStreamSearchInfo</td>
<td>Return info for matching streams</td>
</tr>
<tr>
<td>ATM_TRIGGER_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMTriggerInfo</td>
<td>ATM-1: Get trigger count and time</td>
</tr>
<tr>
<td>ATM_VCC_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMVCCInfo</td>
<td>Get per VCC counts</td>
</tr>
<tr>
<td>ATM_VCDB_LIST_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMVCDBInfo</td>
<td>Get the VC DataBase info</td>
</tr>
</tbody>
</table>

### ATM - HTSetCommand Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM_CLIP_ESTABLISH_CLIENT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMClipEstablishClient</td>
<td>Reset the signaling test</td>
</tr>
<tr>
<td>ATM_CLIP_RELEASE_CLIENT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMClipReleaseClient</td>
<td>Reset the signaling test</td>
</tr>
<tr>
<td>ATM_CONN_PARAMS_COMPLETE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMConnParamsComplete</td>
<td>De-allocates the &quot;Source&quot; Frame stored in memory</td>
</tr>
<tr>
<td>ATM_CONN_PARAMS_RESET</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMConnParamsReset</td>
<td>Deregister an ATM address</td>
</tr>
<tr>
<td>ATM_FRAME_CLEAR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMFrameClear</td>
<td>Register an ATM address</td>
</tr>
<tr>
<td>ATM_ILMI_DEREGISTER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMILMIDeregister</td>
<td>Establish the SAAL at interface</td>
</tr>
<tr>
<td>ATM_ILMI_REGISTER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMILMIRegister</td>
<td>Release the SAAL at interface</td>
</tr>
<tr>
<td>ATM_SAAL_ESTABLISH</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSAALEstablish</td>
<td>Release all calls from Emulator</td>
</tr>
<tr>
<td>ATM_SAAL_RELEASE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSAALRelease</td>
<td>Start a signaling test</td>
</tr>
<tr>
<td>ATM_SIG_EMUL_RESET</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ATMSigEmulReset</td>
<td>Stop a signaling test</td>
</tr>
<tr>
<td>ATM_START_SETUP</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMStartSetup</td>
<td></td>
</tr>
<tr>
<td>ATM_STOP_SETUP</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>ATMStopSetup</td>
<td></td>
</tr>
</tbody>
</table>
## ATM - HTSetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>iType1</td>
<td>ATM_CALL_ADDR_LIST</td>
<td>int HTSetStructure(ATM_CALL_ADDR_LIST, 0, 0, 0, (void*)pATMCallAddrList, sizeof(ATMCallAddrList), iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

### Related Structure

ATMCallAddrList

This structure is used to define the called ATM addresses for a signaling test.

```c
typedef struct tagATMCallAddrList {
    unsigned short uiStartAddrIndex; /* Index of first element */
    unsigned short uiCount;         /* Number of array elements */
    ATMAddress atmAddress[ATM_MAX_CALL_ADDRESSES]; /* List of addresses to load */
} ATMCallAddrList;
```

### Comment

The ATMCallAddrList structure is used to define the called ATM addresses for a signaling test.

## ATM - HTSetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>iType1</td>
<td>ATM_CALL_SETUP</td>
<td>int HTSetStructure(ATM_CALL_SETUP, 0, 0, 0, (void*)pATMCallSetupParams, sizeof(ATMCallSetupParams), iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

### Related Structure

ATMCallSetupParams

The ATMCallSetupParams structure defines what is signaled by the signaling test when attempting to make a call setup. This structure is only used for the signaling test and should not be used when attempting to establish an SVC over which data will actually be sent.
typedef struct tagATMCallSetupParams
{
    unsigned short    uiCallSetupIndex;    /* The index of the machine */

    /* ATM Traffic Descriptor Information Element */
    unsigned long   ulFwdTrafficDescriptorType;  /* one of ATM_TD from below */
    unsigned long   ulFwdPCR_0;                /* Peak cell rate 0 in cells/sec */
    unsigned long   ulFwdPCR_01;               /* Peak cell rate 0+1 in cells/sec */
    unsigned long   ulFwdSCR_0;                /* Sus cell rate 0 in cells/sec */
    unsigned long   ulFwdSCR_01;               /* Sus cell rate 0+1 in cells/sec */
    unsigned long   ulFwdMBS_0;                /* Maximum Burst Size 0 in cells */
    unsigned long   ulFwdMBS_01;               /* Maximum Burst Size 0+1 in cells */

    /* Quality of Service Information Element */
    unsigned char   ucFwdQOS;      /* Forward - one of ATM_QOS values */
    unsigned char   ucBwdQOS;      /* Backward - one of ATM_QOS values */

    /* Broadband Bearer Capability Information Element */
    unsigned char   ucBbcClass;         /* One of ATM_B_BC_CLASS values */
    unsigned char   ucBbcTimingReq;   /* One of ATM_B_BC timing values */
    unsigned char   ucBbcTrafficType;   /* One of ATM_B_BC_TYPE values */
    unsigned char   ucBbcSusceptibleToClipping;   /* ATM_B_BC clipping values*/
} ATMCallSetupParams;

Comment

iType1

ATM_CLASSICAL_IP

Description
Configure CLIP ARP server info

Usage
int HTSetStructure(ATM_CLASSICAL_IP,
                   0, 0, 0,
                   (void*)pATMClassicalIP,
                   sizeof(ATMClassicalIP),
                   iHub, iSlot, iPort);

Related Structure
ATMClassicalIP

This structure provides the user with all of the information necessary to work with a CLIP ARP server. The uiInterCallGap specifies the time between successive data direct VCC connection attempts to another IP client. This can be an issue when a single ARP server response is capable of satisfying multiple ARP requests. Rather than bursting all of the connections to the destination client at once, the call gap allows the user to slow down the rate of connection. If all connections should be connected as quickly as possible, then there is nothing that precludes this value from being 0.

typedef struct tagATMClassicalIP
{
    unsigned char   ucArpServerAtmAddr[20];    /* The server's ATM address */
    unsigned char   ucArpClientIpAddr[4];      /* The card's IP address */
    unsigned long   uiInterArpGap;             /* millisec between ARP retries*/
    unsigned long   uiInterCallGap;            /* millisec between data conn's*/
    unsigned short  uiArpRetries;              /* Max number of ARP retries */
    /* RFC1577_PARAMS */
} ATMClassicalIP;

Comment
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_COPY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Copy connections for sigtest</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetStructure(ATM_CONN_COPY, 0, 0, 0, (void*)pATMConnectionCopyParams, sizeof(ATMConnectionCopyParams), iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

### Related Structure

**ATMConnectionCopyParams**

The copy params structure allows a sequence of connections already configured on the target card to be copied to another range of connections. This structure allows the user to copy an existing range of existing connections to a new range of connections in preparation for a signaling test. This parameter is used to bypass the lengthy downloads which would result from sending all of the duplicated connection data over the serial port. This structure specifies a source connection index which specifies the first connection to be copied. This structure defines the first target destination index of the copy. Finally, the structure includes the desired number of connections to copy. The actual number of copies which will be made may be reduced from the desired if a connection is attempted to be used as both a source and a destination within the same command.

**Example 1: ("Normal" operation)**

Source = 1, Dest = 5, Count = 3

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Configuration</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>H</td>
</tr>
</tbody>
</table>

**Example 2: ("Incorrect" operation)**

Source = 1, Dest = 3, Count = 4

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Configuration</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>H</td>
</tr>
</tbody>
</table>

Example 2 fails when the source of a copy is the same as the destination address of the first copy (3). In this case only two connections were copied instead of the requested 4.

```c
typedef struct tagATMConnectionCopyParams
{
    unsigned long ulSrcIndex; /* The starting source range index */
    unsigned long ulDstIndex; /* The starting dest range index */
    unsigned long ulCount; /* The number of conns to copy */
} ATMConnectionCopyParams;
```

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_MODIFY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Fill field of conn. for sigtest</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ATM_CONN_MODIFY, 0, 0, 0, (void*)pATMConnectionModify, sizeof(ATMConnectionModify), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMConnectionModify</td>
</tr>
</tbody>
</table>

The modify connections structure allows a sequence of connections already configured on the target card to have one of their fields modified with successively increasing values. This structure will modify a particular field of ulCount successive connections with a constantly changing value. The parameter identified in ulType indicates which field of the connection structure is of interest. The identified field of the connection at ulSrcIndex is taken to be the starting value for the modify. The value is increased by delta and the new value is written into the next connection (connection number ulSrcIndex+1). The new value is increased again by delta and written to the next connection. This process repeats until ulCount connections have been written with a new value. The field of interest is identified by one of the ATM_CALL_PARAM defines below.

For Example:
Source = 3, Count = 3, Delta = 2

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>58</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

typedef struct tagATMConnectionModify {
  unsigned long  ulSrcIndex;  /* Conn holds starting field value */
  unsigned long  ulCount;     /* Number of conns to change */
  unsigned long  ulType;      /* Identifies which field to change */
  long           lDelta;       /* Successive increment value */
} ATMConnectionModify;
### `ATM_CONN_MODIFY_ARRAY`

**Description**: Mod field of conn. for sigtest

**Usage**
```c
int HTSetStructure(ATM_CONN_MODIFY_ARRAY, 0, 0, 0, (void*)pATMConnectionModifyArray, sizeof(ATMConnectionModifyArray), iHub, iSlot, iPort);
```

**Related Structure**: `ATMConnectionModifyArray`

The modify connections array structure allows a sequence of connections already configured on the target card to have one of their fields modified with user-supplied data.

This structure will modify a particular field of `ulCount` successive connections with user-specified values. The parameter identified in `ulID` indicates which field of the connection structure is to be modified. `ulSourceIndex` specifies the index of the first connection which is modified. The `ulValues` array contains the values which will be placed into the successive connections.

For Example:
```
Count = 3, SrcIndex = 2, Values = (6, 5, 4)
```

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>58</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

```c
typedef struct tagATMConnectionModifyArray
{
    unsigned long    ulSrcIndex;
    unsigned long    ulCount;
    unsigned long    ulID;
    unsigned long    ulArraySize; /* reserved - Currently it is the same as ulCount */
    unsigned long    ulValues[ATM_MAX_ARRAY_DIM];
} ATMConnectionModifyArray;
```

### `ATM_CONN_TRIGGER_PARAMS`

**Description**: Define per conn trigger event

**Usage**
```c
int HTSetStructure(ATM_CONN_TRIGGER_PARAMS, 0, 0, 0, (void*)pATMConnTriggerParams, sizeof(ATMConnTriggerParams), iHub, iSlot, iPort);
```

**Related Structure**: `ATMConnTriggerParams`

`ATMConnTriggerParams` allows the user to define a trigger event pattern to be used for a particular connection. The per connection trigger is associated with comparator one.

```c
typedef struct tagATMConnTriggerParams
{
    unsigned short    uiConnIndex; /* Connection to test */
    unsigned long    ulComp1Pattern; /* Pattern to test for per conn */
} ATMConnTriggerParams;
```
iType1 | ATM_CONNECTION
---|---
Description | Define a connection for sigtest
Usage | int HTSetStructure(ATM_CONNECTION, 0, 0, 0, (void*)pATMConnection, sizeof(ATMConnection), iHub, iSlot, iPort);

Related Structure | ATMConnection

Defines the content of a connection for the signaling test. This structure assumes that the ATM address list and the call parameters for the signaling test have already been defined. Essentially, there are three types of signaling tests which can be run: a smooth test, a bursty test, and a random distribution test. A signaling test works by creating a series of machines which will attempt to establish a connection and teardown a certain number of times as defined by ulCallCountLimit. The parameters at the top of the structure define what is signaled to the DUT when a call connection is attempted. The "stop on error" field allows this machine to quit attempting further calls if one of its previous calls were released by the switch while a call was proceeding or after a call was established. There are three types of initiating timings as follows:

SMOOTH: The time from a call teardown to a call initiation is constant and defined by the ulInterCallGap.

BURSTY: The time from a call teardown to a call initiation is initially defined by the ulBurstGap for the first ulBurstCount calls and then defined by the ulInterCallGap for the remaining calls.

RANDOM: The time from a call teardown to a call initiation is calculated between each call to be ulInterCallGap + random(ulInterCallGapDelta).

There are also three types of completion timings as follows:

CONST: The time from a call connection to initiating a call teardown is constant and defined by ulCallLength. Calls will be torn down at this time independent of their current connection state.

CONST_FROM_SETUP: The time from a call setup to initiating a call teardown is constant and defined by ulCallLength. Calls which are not complete at this point in time will be allowed to complete before a new call is initiated.

RANDOM: The time from a call connection to initiating a call teardown is calculated for each call to be ulCallLength + random(ulRandomLengthDelta). Calls will be torn down at this time independent of their current connection state.
typedef struct tagATMConnection
{
    unsigned long ulIndex; /* Connection Number */
    unsigned char ucType;  /* PVC or SVC */
    unsigned char ucRateClass; /* CBR, VBR, or UBR */
    unsigned long ulRatePCR; /* CBR, VBR, UBR Peak Cell Rate */
    unsigned long ulRateSCR; /* VBR Sustainable Cell Rate */
    unsigned long ulRateMBS; /* VBR Max Burst Size */
    ATMCellTime ctCellDelayVar; /* CBR, VBR Cell Delay Variation */
    unsigned long ulCellHeader; /* Cell header - for GFC, PT, CLP */
    unsigned short uiCallSetupIndex; /* Corresponding call info index */
    unsigned short uiAddressIndex; /* Corresponding ATM addr. index */
    unsigned char ucCallDistType; /* One of ATM_CALL_DIST types */
    unsigned char ucCallLengthType; /* One of ATM_CALL_LENGTH types */
    unsigned char ucStopOnError; /* Stop on unexpected call release */
    unsigned char ucLogEvents; /* Set to 0 */
    unsigned char ucEnableCellLoadGen; /* Set to 0 */
    unsigned long ulCallStartDelay; /* Time from start to first call */
    unsigned long ulCallCountLimit; /* The no. of calls to attempt */
    unsigned long ulCallLength; /* Minimum wait before teardown */
    unsigned long ulRandomLengthDelta; /* The time variation */
    unsigned long ulInterCallGap; /* Min wait before initiation */
    unsigned long ulRandomGapDelta; /* The time variation */
    unsigned long ulBurstCount; /* Number of calls to burst */
    unsigned long ulInterBurstGap; /* Min wait before teardown */
} ATMConnection;
**iType1**: ATM_DS1_E1_LINE_PARAM

**Description**: Config ATM and PHY for DS1 or E1

**Usage**: int HTSetStructure(ATM_DS1_E1_LINE_PARAM, 0, 0, 0, (void*)pATMDS1E1LineParams, sizeof(ATMDS1E1LineParams), iHub, iSlot, iPort);

**Related Structure**: ATMDS1E1LineParams

The ATMDS1E1LineParams structure is used to configure the physical layer and the ATM layer for the following cards: AT-9015, AT-9020. The following values are valid for each card:

**AT-9015**:
- `ucFramingMode` - ATM_DS1_CELL_FRAMING or ATM_DS1_PLCP_FRAMING
- `ucTxClockSource` - ATM_INTERNAL_CLOCK or ATM_LOOP_TIMED_CLOCK
- `ucCellScrambling` - 0 (disabled) or 1 (enabled)
- `ucRxErroredCells` - ATM_DROPErroRED CELLS or ATM_CORRECTErroRED CELLS or ATM_RXErroRED CELLS
- `ucLoopBackEnable` - ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- `ucLineBuildout` - ATM_DS1_0_TO_133_BUILDOUT or ATM_DS1_133_TO_266_BUILDOUT or ATM_DS1_266_TO_399_BUILDOUT or ATM_DS1_399_TO_655_BUILDOUT or ATM_DS1_N7X5_DB_BUILDOUT or ATM_DS1_N15_DB_BUILDOUT or ATM_DS1_N22X5_DB_BUILDOUT
- `ucLineCoding` - ATM_DS1_AMI_ENCODING or ATM_DS1_B8ZS_ENCODING
- `ucLineFraming` - ATM_DS1_D4_LINE_FRAMING or ATM_DS1_ESF_LINE_FRAMING
- `ucIdleCellHeader` - Contains idle cell header without HEC byte

**AT-9020**:
- `ucFramingMode` - ATM_E1_CELL_FRAMING or ATM_E1_PLCP_FRAMING
- `ucTxClockSource` - ATM_INTERNAL_CLOCK or ATM_LOOP_TIMED_CLOCK
- `ucCellScrambling` - 0 (disabled) or 1 (enabled)
- `ucRxErroredCells` - ATM_DROPErroRED CELLS or ATM_CORRECTErroRED CELLS or ATM_RXErroRED CELLS
- `ucLoopBackEnable` - ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- `ucLineBuildout` - ATM_E1_BUILDOUT or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- `ucLineCoding` - ATM_E1_AMI_ENCODING or ATM_E1_HDB3_ENCODING
- `ucLineFraming` - 0
- `ucIdleCellHeader` - Contains idle cell header without HEC byte

typedef struct tagATMDS1E1LineParams
{
    /* Interface control flags */
    unsigned char ucFramingMode; /* Physical layer framing */
    unsigned char ucTxClockSource; /* Transmit clock source */
    unsigned char ucCellScrambling; /* Cell Payload Scrambling */
    unsigned char ucHecCoset; /* HEC Coset Usage */
    unsigned char ucRxErroredCells; /* Receive HEC error handling */
    unsigned char ucLoopbackEnable; /* Loopback mode */
    unsigned char ucLineBuildout; /* Pulse shaping for PHY */
    unsigned char ucLineCoding; /* Line symbol coding */
    unsigned char ucLineFraming; /* Framing format for PHY */

    /* Idle cell definition */
    unsigned char ucIdleCellHeader[4]; /* Idle cell value */
} ATMDS1E1LineParams;

**Comment**
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>ATM_DS3_E3_LINE_PARAM</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Config ATM and PHY for DS3 or E3</td>
</tr>
</tbody>
</table>
| **Usage** | \[
int HTSetStructure(ATM_DS3_E3_LINE_PARAM, 0, 0, 0, (void*)pATMDS3E3LineParams, sizeof(ATMDS3E3LineParams), iHub, iSlot, iPort);
\] |
| **Related Structure** | ATMD3E3LineParams |

The ATMD3E3LineParams structure is used to configure the physical layer and the ATM layer for the following cards: AT-9034, AT-9045. The following values are valid for each card:

**AT-9034:**
- **ucFramingMode** - ATM_E3_CELL_FRAMING or ATM_E3_PLCP_FRAMING
- **ucTxClockSource** - ATM_INTERNAL_CLOCK or ATM_LOOP_TIMED_CLOCK
- **ucCellScrambling** - 0 (disabled) or 1 (enabled)
- **ucHecCoset** - 0 (disabled) or 1 (enabled)
- **ucRxErroredCells** - ATM_DROP_ERRORED_CELLS or ATM_CORRECT_ERRORED_CELLS or ATM_RX_ERRORED_CELLS
- **ucLoopbackEnable** - ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- **ucLineBuildout** - 0
- **ucIdleCellHeader** - Contains idle cell header without HEC byte

**AT-9045:**
- **ucFramingMode** - ATM_DS3_CELL_FRAMING or ATM_DS3_PLCP_FRAMING
- **ucTxClockSource** - ATM_INTERNAL_CLOCK or ATM_LOOP_TIMED_CLOCK
- **ucCellScrambling** - 0 (disabled) or 1 (enabled)
- **ucHecCoset** - 0 (disabled) or 1 (enabled)
- **ucRxErroredCells** - ATM_DROP_ERRORED_CELLS or ATM_CORRECT_ERRORED_CELLS or ATM_RX_ERRORED_CELLS
- **ucLoopbackEnable** - ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- **ucLineBuildout** - ATM_DS3_SHORT_BUILDOUT or ATM_DS3_LONG_BUILDOUT
- **ucIdleCellHeader** - Contains idle cell header without HEC byte

```c
typedef struct tagATMDS3E3LineParams
{
    /* Line interface control flags */
    unsigned char ucFramingMode; /* Physical layer framing */
    unsigned char ucTxClockSource; /* Transmit clock source */
    unsigned char ucCellScrambling; /* Cell Payload Scrambling */
    unsigned char ucHecCoset; /* HEC Coset Usage */
    unsigned char ucRxErroredCells; /* Receive HEC error handling */
    unsigned char ucLoopbackEnable; /* Loopback mode */
    unsigned char ucLineBuildout; /* Pulse shaping for PHY */
    /* Idle cell definition */
    unsigned char ucIdleCellHeader[4]; /* Idle cell value */
} ATMDS3E3LineParams;
```
<table>
<thead>
<tr>
<th>iTyper1</th>
<th>ATM_ELAN_DEREGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Deregister an existing LEC</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ATM_ELAN_DEREGISTER, 0, 0, 0, (void*)pATMELANDeregister, sizeof(ATMELANDeregister), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMELANDeregister</td>
</tr>
<tr>
<td>Allows the user to deregister from a specific ELAN.</td>
<td></td>
</tr>
<tr>
<td>typedef struct tagATMELANDeregister</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucInstance; /* Identifies which ELAN */</td>
<td></td>
</tr>
<tr>
<td>} ATMELANDeregister;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iTyper1</th>
<th>ATM_ELAN_REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Define and register a LEC</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ATM_ELAN_REGISTER, 0, 0, 0, (void*)pATMELANRegister, sizeof(ATMELANRegister), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMELANRegister</td>
</tr>
<tr>
<td>typedef struct tagATMELANRegister</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucInstance; /* Identifies which ELAN */</td>
<td></td>
</tr>
<tr>
<td>*/ Join parameters */</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucInitMethod; /* One of ATM_ELAN_INIT from below */</td>
<td></td>
</tr>
<tr>
<td>*/ manual ATM address for direct joins */</td>
<td></td>
</tr>
<tr>
<td>ATMAddress ManualAtmAddr; /* LES or LECS address if required */</td>
<td></td>
</tr>
<tr>
<td>*/ LEC parameters */</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucC2Type; /* One of ATM_ELAN_TYPE */</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucC3Mtu; /* Max Transfer Unit, One of ATM_ELAN_MTU */</td>
<td></td>
</tr>
<tr>
<td>*/ Target ELAN name (may be NULLified) */</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucC5Name[ATM_MAX_ELAN_NAME];</td>
<td></td>
</tr>
<tr>
<td>*/ Base MAC address and count */</td>
<td></td>
</tr>
<tr>
<td>unsigned char ucC6MacAddr[6]; /* ELAN MAC address */</td>
<td></td>
</tr>
<tr>
<td>unsigned short uiC7ControlTimeout; /* Timeout in seconds, control conn */</td>
<td></td>
</tr>
<tr>
<td>unsigned short uiC13ArpRetryCount;</td>
<td></td>
</tr>
<tr>
<td>unsigned short uiC20ArpResponseTime; /* Max wait time for ARP response */</td>
<td></td>
</tr>
<tr>
<td>} ATMELANRegister;</td>
<td></td>
</tr>
</tbody>
</table>

Comment
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>ATM_FRAME_COPY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Copy Frame params to a given # of streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ATM_FRAME_COPY, 0, 0, 0, (void*)pATMFrameCopyReq, sizeof(ATMFrameCopyReq), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td><strong>ATMFrameCopyReq</strong></td>
</tr>
</tbody>
</table>

```c
typedef struct tagATMFrameCopyReq {
    unsigned short uiStartStrNum; /* Stream index of the first stream to be copied to. */
    unsigned short uiStrCount; /* The number of streams, which the frame, stored in memory, is used as a basis for modification. */
    unsigned short uiNumMods; /* The number of FrameCopyMod structures which immediately follow this structure. */
    ATMFrameCopyMod ModArray[ATM_MAX_MODS];
} ATMFrameCopyReq;
```

**Comment**

Supported by ATM SmartCard version 2.0 and later.
### iType1: Structure Reference

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_FRAME_DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Config a new or existing frame</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ATM_FRAME_DEF, 0, 0, 0, (void*)pATMFrameDefinition, sizeof(ATMFrameDefinition), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

#### Related Structure: ATMFrameDefinition

This structure allows the user to define a frame's contents and to associate a frame with a particular stream. When this data is sent to the card the frame definition will fail if the stream index is greater than the maximum number of streams supported by our card or if the stream to which the frame will be bound has not been defined yet or if the associated stream does not exist and the connection is open. The frame length defines the length of the user data to be segmented in the aal5 layer. The custom frame data is placed into the frame first. If there is still more data required to complete the frame then the remaining bytes of the frame are filled with the fill pattern. This data structure is intended to be used with the frame length set equal to or greater than the data length. If the data length is greater than the frame length then only the first frame length bytes of the custom data are used in the definition of the frame.

For Example:

```c
uiDataLength = 10
uiFrameLength = 14
ucFrameData[] = 0x0102030405060708090A0B0C0D....
uiFrameFillPattern = 0xABCD
The defined frame becomes: 0x0102030405060708090AABCDABCD
```

Example 2:

```c
uiDataLength = 10
uiFrameLength = 8
ucFrameData[] = 0x0102030405060708090A0B0C0D....
uiFrameFillPattern = 0xABCD
The defined frame becomes: 0x0102030405060708
```

define struct tagATMFrameDefinition
{
    unsigned short   uiStreamIndex;   /* Stream identifier/index            */
    unsigned short   uiFrameLength;   /* The total length of the frame      */
    unsigned short   uiDataLength;   /* The length of the custom data      */
    unsigned short   uiFrameFillPattern;   /* Fill pattern word            */
    unsigned long   ulFrameFlags;   /* Special frame generation flags */
    unsigned char   ucFrameData[2048];   /* Custom frame data            */
} ATMFrameDefinition;

### Comment
**iType1** ATMGLOBAL_TRIGGER_PARAMS

**Description**
Define global trigger event

**Usage**
```c
int HTSetStructure(ATM_GLOBAL_TRIGGER_PARAMS,
                   0, 0, 0,
                   (void*)pATMGlobalTrigger,
                   sizeof(ATMGlobalTrigger),
                   iHub, iSlot, iPort);
```

**Related Structure** ATMGlobalTrigger

The ATMGlobalTrigger structure defines global trigger conditions which apply to all received frames. In ATM-2 the triggers are only active on the receive side. Each VCC has its own latency and trigger counter to facilitate measurement of per connection statistics. There are two events which can be defined. The first event is associated with comparator two and is a global event. This occurrence of this event is tested against all frames received on all connections. The second event is defined per connection. There is no mask, but each connection may specify a pattern which will be used to test frames received only on that connection.

```c
typedef struct tagATMGlobalTrigger
{
  /* Trigger control */
  unsigned char ucEnable;       /* Enable triggers */
  unsigned char ucCompCombo;    /* Comparison combination */

  /* Global Trigger parameters */
  unsigned long ulComp2Pattern; /* Global event pattern */
  unsigned short uiComp1Offset; /* Per conn triggers' pattern offset*/
  unsigned short uiComp2Offset; /* Global trigger pattern offset */
  unsigned long ulComp1Mask;    /* Mask bits for per conn event */
  unsigned long ulComp2Mask;    /* Mask bits for global event */
} ATMGlobalTrigger;
```

**Comment**

---

**iType1** ATM_ILMI

**Description**
Set ILMI timers and ESI

**Usage**
```c
int HTSetStructure(ATM_ILMI,
                   0, 0, 0,
                   (void*)pATMILMIParams,
                   sizeof(ATMILMIParams),
                   iHub, iSlot, iPort);
```

**Related Structure** ATMILMIParams

The ATMILMIParams structure is used to indicate to the card what the desired end system identifier is and what the proper timeout values are for ILMI. These are the values which are used by ILMI to identify to the network what the ATM address of the card will be.

```c
typedef struct tagATMILMIParams
{
  unsigned long ulColdStartTimer;    /* Timeout time in .01 S resolution */
  unsigned long ulRegTimeoutTimer;   /* Timeout time in .01 S resolution */
  unsigned char ucESI[6];           /* End System Identifier */
} ATMILMIParams;
```

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_ILMI_STATIC_REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Force an ATM address statically</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(ATM_ILMI_STATIC_REGISTER, 0, 0, 0, (void*)pATMILMIStruct, sizeof(ATMILMIStruct), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td>Related Structure</td>
<td>ATMLMIStruct</td>
</tr>
<tr>
<td>Description</td>
<td>The following structure is used to specify to the firmware a particular ATM address to be used. This function is most frequently used when connecting to management entities within a switch.</td>
</tr>
<tr>
<td>Related Structure</td>
<td>ATMLMIStruct</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

```c
typedef struct tagATMILMIStructParams
{
    ATMAddress atmAddress;
} ATMLMIStructParams;
```

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_INCOMING_SVC_METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Incoming SVC method</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(ATM_INCOMING_SVC_METHOD, 0, 0, 0, (void*)pATMIncomingSVCMethod, sizeof(ATMIncomingSVCMethod), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td>Related Structure</td>
<td>ATMIncomingSVCMethod</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

```c
typedef struct tagATMIncomingSVCMethod
{
    unsigned char ucReserved[3];
    unsigned char ucMethod;        /* Use one of the following defined below */
} ATMIncomingSVCMethod;
```
**iType1**  | **ATM_LINE**  
---|---  
**Description** | Physical and ATM layer config  
**Usage** | int HTSetStructure(ATM_LINE, 0, 0, 0, (void*)pATMLineParams, sizeof(ATMLineParams), iHub, iSlot, iPort);  
**Related Structure** | ATMLineParams  

The ATMLineParams structure is used to configure the physical layer and the ATM layer for the following cards: AT-9622, AT-9155, AT-9025. The following values are valid for each card:

**AT-9622:**
- `ucFramingMode`: ATM_OC12_FRAMING or ATM_STM4_FRAMING
- `ucTxClockSource`: ATM_INTERNAL_CLOCK or ATM_LOOP_TIMED_CLOCK
- `ucCellScrambling`: 0 (disabled) or 1 (enabled)
- `ucRxErroredCells`: ATM_DROPErrored_CELLS or ATM_CORRECTErrored_CELLS or ATM_RXErrored_CELLS
- `ucLoopBackEnable`: ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- `ucIdleCellHeader`: Contains idle cell header without HEC byte

**AT-9155:**
- `ucFramingMode`: ATM_OC3_FRAMING or ATM_STM1_FRAMING
- `ucTxClockSource`: ATM_INTERNAL_CLOCK or ATM_LOOP_TIMED_CLOCK
- `ucCellScrambling`: 0 (disabled) or 1 (enabled)
- `ucRxErroredCells`: ATM_DROPErrored_CELLS or ATM_CORRECTErrored_CELLS or ATM_RXErrored_CELLS
- `ucLoopBackEnable`: ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- `ucIdleCellHeader`: Contains idle cell header without HEC byte

**AT-9025:**
- `ucFramingMode`: ATM_25_FRAMING
- `ucTxClockSource`: ATM_INTERNAL_CLOCK
- `ucCellScrambling`: 1 (enabled)
- `ucRxErroredCells`: ATM_DROPErrored_CELLS or ATM_RXErrored_CELLS
- `ucLoopBackEnable`: ATM_LOOPBACK_DISABLED or ATM_LOOPBACK_LOCAL_PHY or ATM_LOOPBACK_REMOTE_PHY
- `ucIdleCellHeader`: Contains idle cell header without HEC byte

```c
typedef struct tagATMLineParams
{
    /* Line interface control flags */
    unsigned char ucFramingMode;     /* Physical layer framing */
    unsigned char ucTxClockSource;   /* Transmit clock source */
    unsigned char ucCellScrambling;  /* Cell Payload Scrambling */
    unsigned char ucHecCoset;        /* HEC Coset Usage */
    unsigned char ucRxErroredCells;  /* Receive HEC error handling */
    unsigned char ucLoopbackEnable;  /* Loopback mode */

    /* Idle cell definition */
    unsigned char ucIdleCellHeader[4]; /* Idle cell value */
} ATMLineParams;
```
### iType1  
**ATM_PER_CONN_BURST**

<table>
<thead>
<tr>
<th>Description</th>
<th>Burst count per connection</th>
</tr>
</thead>
</table>
| **Usage**   | int HTSetStructure(ATM_PER_CONN_BURST,  
|             |   <index>, <count>, 0,   
|             |   (void*)pATMPerConnBurstCount, 
|             |   sizeof(ATMPerConnBurstCount),  
|             |   iHub, iSlot, iPort) ;       |
| **Related Structure** | ATMPerConnBurstCount |
|              | typedef struct    tagATMPerConnBurstCount  
|              | { unsigned short   uiStartConnIdx;   
|              |   unsigned short   uiConnCount;   
|              |   unsigned char   ucFunction;         /* Use one of the following defined below */ 
|              |   unsigned char   ucReserved[3];   
|              |   unsigned long   ulReserved;   
|              |   unsigned long   ulFrameBurstSize;   /* from 1 to 2,097,151 */ 
|              | ) ATMPerConnBurstCount;            |
| **Comment** | Supported by ATM SmartCard version 2.0 and later. |

### iType1  
**ATM_PER_PORT_BURST**

<table>
<thead>
<tr>
<th>Description</th>
<th>Burst count per port</th>
</tr>
</thead>
</table>
| **Usage**   | int HTSetStructure(ATM_PER_PORT_BURST,  
|             |   0, 0, 0,   
|             |   (void*)pATMPerPortBurstCount, 
|             |   sizeof(ATMPerPortBurstCount),  
|             |   iHub, iSlot, iPort) ;       |
| **Related Structure** | ATMPerPortBurstCount |
|              | typedef struct    tagATMPerPortBurstCount  
|              | { unsigned char   ucFunction;         /* Use one of the following defined below */ 
|              |   unsigned char   ucReserved[3];   
|              |   unsigned long   ulReserved;   
|              |   unsigned long   ulFrameBurstSize;   /* from 1 to 4,294,967,295 ( all 32-bits ) */ 
|              | ) ATMPerPortBurstCount;            |
| **Comment** | Supported by ATM SmartCard version 2.0 and later. |
### ATM_SCHED_PARAMS

**Description**  
Cell-scheduling method for transmission

**Usage**  
```c
int HTSetStructure(ATM_SCHED_PARAMS, 0, 0, 0, (void*)pATMSchedParams, sizeof(ATM_SCHED_PARAMS), iHub, iSlot, iPort);
```

**Related Structure**  
ATMSchedParams

```c
typedef struct tagATMSchedParams
{
    unsigned long ulUtilization; /* The amount of bandwidth as a percent of the total line rate. Percent can range from 1 -100, inclusive, in steps of 1. This does not apply to the Early Sched. Type */
    unsigned short uiSchedType; /* Determines the type of scheduling used the next time streams are scheduled. Use one of the following defined below. */
    unsigned short uiReserved;
    unsigned long ulReserved1;
    unsigned long ulReserved2;
} ATMSchedParams;
```

**Comment**  
Supported by ATM SmartCard version 2.0 and later.

---

### ATM_SSCOP

**Description**  
Set SSCOP timers and config

**Usage**  
```c
int HTSetStructure(ATM_SSCOP, 0, 0, 0, (void*)pATMSSCOPParams, sizeof(ATM_SSCOPParams), iHub, iSlot, iPort);
```

**Related Structure**  
ATMSSCOPParams

Defines Maximum and Timer values for use in configuring the SSCOP level of the ATM interface.

```c
typedef struct tagATMSSCOPParams
{
    unsigned long ulMaxCC;
    unsigned long ulMaxPD;
    unsigned long ulMaxStat;
    unsigned long ulMaxReseq;
    unsigned long ulMaxWindow;
    unsigned long ulTmrCC;
    unsigned long ulTmrKeepAlive;
    unsigned long ulTmrNoResp;
    unsigned long ulTmrPoll;
    unsigned long ulTmrIdle;
} ATMSSCOPParams;
```
<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>ATM_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Config a new or existing stream</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ATM_STREAM, 0, 0, 0, (void*)pATMStream, sizeof(ATMStream), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMStream</td>
</tr>
</tbody>
</table>

The ATMStream structure is used to define what is signaled for a data stream and what the performance of the data stream will be once it is established.
typedef struct tagATMStream
{
    unsigned short   uiIndex;         /* Stream Index                   */
    unsigned char   ucConnType;      /* use Connection type* defines   */
    STR_CONN_TYPE_PVC
    STR_CONN_TYPE_SVC
    STR_CONN_TYPE_INCOMING_SVC
    unsigned char   ucEncapType;   /* use "encapsulation type" defines */
    STR_ENCAP_TYPE_NULL
    STR_ENCAP_TYPE_LANE_802_3
    STR_ENCAP_TYPE_LANE_802_5
    STR_ENCAP_TYPE_RFC1577
    unsigned char   ucGenRateClass; /* use "rate class" defines          */
    STR_RATE_CLASS_CBR
    STR_RATE_CLASS_VBR
    STR_RATE_CLASS_UBR
    unsigned long   ulGenPCR;      /* Peak Cell Rate (cells/sec)     */
    unsigned long   ulGenSCR;      /* Sustainable Cell Rate (cells/sec)*/
    unsigned long   ulGenMBS;      /* Maximum Burst Size (cells)       */
    unsigned long   ulCellTime;    /* Cell Delay Variation Tolerance(Time)*/
    unsigned char   ucFwdTdType;   /* one of ATM_TD values (Traffic Desc) */
    unsigned long   ulFwdPCR0;      /* PCR 0 rate (cells/sec)         */
    unsigned long   ulFwdPCR01;      /* PCR 0+1 rate (cells/sec)      */
    unsigned long   ulFwdSCR0;      /* SCR 0 rate (cells/sec)         */
    unsigned long   ulFwdSCR01;      /* SCR 0+1 rate (cells/sec)      */
    unsigned long   ulFwdMBS0;      /* MBS 0 (cells)                  */
    unsigned long   ulFwdMBS01;      /* MBS 0+1 (cells)               */
    unsigned char   ucBwdTdType;   /* one of ATM_TD values (Traffic Desc) */
    unsigned long   ulBwdPCR0;      /* PCR 0 rate (cells/sec)         */
    unsigned long   ulBwdPCR01;      /* PCR 0+1 rate (cells/sec)      */
    unsigned long   ulBwdSCR0;      /* SCR 0 rate (cells/sec)         */
    unsigned long   ulBwdSCR01;      /* SCR 0+1 rate (cells/sec)      */
    unsigned long   ulBwdMBS0;      /* MBS 0 (cells)                  */
    unsigned long   ulBwdMBS01;      /* MBS 0+1 (cells)               */
    unsigned char   ucFwdQoS;      /* Forward - one of ATM_QOS values */
    unsigned char   ucBwdQoS;      /* Backward - one of ATM_QOS values */
    unsigned char   ucBcClass;      /* One of ATM_B_BC_CLASS values   */
    unsigned char   ucTimingReq;   /* One of ATM_B_BC timing values   */
    unsigned char   ucTrafficType;   /* One of ATM_B_BC_TYPE values   */
    unsigned char   ucClipping;      /* One of ATM_B_BC clipping values      */
    unsigned long   ulCellHeader;   /* Specified for PVC, ignored for SVC */
    unsigned long   ulDestAtmAddr[20];/* Specified for SVC, ignored PVC*/
    unsigned long   ulDestMacAddr[6]; /* Specified for LANE or ignored   */
    unsigned long   ulDestIpAddr[4]; /* Specified for CLIP or ignored   */
    unsigned char   ucSnapHeader[5]; /* Not Used */
    unsigned char   ucElanInst;               /* LANE only 0 to LECs - 1      */
} ATMStream;      /* STREAM_PARAMS; */

Comment

This structure defines the information elements used in signaling a new connection if it is an SVC and the actual generation characteristics of the stream once a connection is established (SVC or PVC). This structure is not intended to be used by a signaling test, but it is intended to be used for all other applications.
### iType1 ATM_STREAM_CONTROL

<table>
<thead>
<tr>
<th>Description</th>
<th>Start, stop, reset, etc a stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>int HTSetStructure(ATM_STREAM_CONTROL, 0, 0, 0, (void*)pATMStreamControl, sizeof(ATMStreamControl), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

#### Related Structure ATMStreamControl

The ATMStreamControl structure allows the user to perform one action on a range of streams. There are five actions which can be performed. The basic life of a stream is as follows: When initializing in a new application it is best to issue a stream ATM_STR_ACTION_DISCONNECT and ATM_STR_ACTION_RESET to all possible streams to remove any previously configured information from the card. Next, a frame is defined and bound to a stream. Then the stream is bound to a connection using an ATM_STR_ACTION_CONNECT. The stream is then started and stopped as appropriate and it may even be disconnected and reconnected a number of times depending upon the application. In the end the stream is stopped and a ATM_STR_ACTION_RESET should be sent to destroy the stream.

```c
typedef struct tagATMStreamControl
{
    unsigned char ucAction;   /* Use one of the following
    ATM_STR_ACTION_RESET       Deletes a stream
    ATM_STR_ACTION_CONNECT     Initiate a conn for stream xmit
    ATM_STR_ACTION_DISCONNECT  Close a conn used for stream xmit
    ATM_STR_ACTION_START         Start xmitting data on open conn
    ATM_STR_ACTION_STOP         Stop xmitting data on open conn */
    unsigned long ulStreamIndex;   /* First stream in the range      */
    unsigned long ulStreamCount;   /* Subsequent stream count         */
} ATMStreamControl;         /*   STREAM_CONTROL_REQ; */
```

### iType1 ATM_STREAM_PARAMS_COPY

<table>
<thead>
<tr>
<th>Description</th>
<th>Copy Stream params to a given # of streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>int HTSetStructure(ATM_STREAM_PARAMS_COPY, 0, 0, 0, (void*)pATMStreamParamsCopy, sizeof(ATMStreamParamsCopy), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

#### Related Structure ATMStreamParamsCopy

```c
typedef struct    tagATMStreamParamsCopy
{
    unsigned short uiSrcStrNum;      /* "Source" stream index. */
    unsigned short uiDstStrNum;      /* Starting stream idx in which the info of the "Source" Stream is to be copied to. */
    unsigned short uiDstStrCount;      /* The number of streams to be created with the "Source" Stream's info. */
    unsigned long ulReserved;        /*   STREAM_CONTROL_REQ; */
} ATMStreamParamsCopy;
```

#### Comment

Supported by ATM SmartCard version 2.0 and later.
### ATM_STREAM_PARAMS_MODIFY

**Description**

`int HTSetStructure (ATM_STREAM_PARAMS_MODIFY
 (<index>, <count>, 0,
 (void*)pATMStreamParamsModify,
 sizeof(ATMStreamParamsModify),
 iHub, iSlot, iPort)`;

**Related Structure**

`typedef struct tagATMStreamParamsModify {
    unsigned short uiStartStrNum;  /* Stream index of the first stream to be modified. */
    unsigned short uiStrCount;     /* The number of streams which are to be modified. */
    unsigned short uiParamItemID;  /* Parameter ID which is to be modified on all requested streams - use one of the following defined below. */
    unsigned short uiParamCount;   /* The number of elements to be specified for the uiParamItemID value. This does not include the parameter size */
    unsigned char ucData[ATM_MAX_ARRAY_DIM]; /* Array contains the number of bytes specified by the uiParamCount*ParamSize */
} ATMStreamParamsModify;`

**Comment**

Supported by ATM SmartCard version 2.0 and later.

---

### ATM_STREAM_PARAMS_FILL

**Description**

`int HTSetStructure (ATMStreamParamsFill, 0, 0, 0,
 (void*)pATMStreamParamsFill,
 sizeof(ATMStreamParamsFill),
 iHub, iSlot, iPort)`;

**Related Structure**

`typedef struct tagATMStreamParamsFill {
    unsigned short uiSrcStrNum;   /* Stream index of the first stream whose value for the desired param is to be used as an initial value for the fill. */
    unsigned short uiDstStrNum;   /* Stream index of the first stream whose param is to be filled. */
    unsigned short uiDstStrCount; /* The number of streams which are to be filled. */
    unsigned short uiParamItemID; /* Parameter ID which is to be filled on all requested streams - use one of the following defined below */
    unsigned char ucDelta[ATM_MAX_DELTA]; /* Successive increment values */
} ATMStreamParamsFill;`

**Comment**

Supported by ATM SmartCard version 2.0 and later.
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>ATM TRIGGER</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Configure triggers in ATM-1</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetStructure(ATM_TRIGGER, 0, 0, 0, (void*)pATMTrigger, sizeof(ATMTrigger), iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

### Related Structure | ATMTrigger
---|---

The ATMTrigger structure defines the events which are used to fire a trigger in ATM-1. There are two types of triggers - frame mode and cell mode. In cell mode each cell is tested to see if it matches the defined event conditions. In cell mode the offset is specified in bytes from the start of the cell. In frame mode the offset is specified in bytes from the start of the ATM cell payload. The header bytes are not included in the offset count at all. Mask values determine which bits are compared in the pattern against the frame. If the bit in the mask is set then the corresponding bit in the pattern is tested. If the bit is zero, then the corresponding bit is ignored. The match field causes the trigger to be fired on either a matching pattern (if match=1) or a non-matching pattern (if match=0).

**ATM-2 Trigger Parameters**
- **Trigger 1 - Per Connection**
- **Trigger 2 - Global**

```c
typedef struct tagATMTrigger {
    /* Trigger control */
    unsigned char ucEnable; /* Enable/Disable triggers */
    unsigned char ucMode; /* Cell or frame level */
    unsigned char ucDirection; /* Transmit or receive */
    unsigned char ucCompCombo; /* One of ATM_COMP values */
    unsigned char ucHeaderNoMatch; /* Event is a non match of header */
    unsigned char ucComp1NoMatch; /* Event is non match of Comp 1 */
    unsigned char ucComp2NoMatch; /* Event is non match of Comp 2 */

    /* Cell header comparator parameters */
    unsigned long ulHeaderPattern; /* GFC/VPI/VCI/PT/CLP pattern */
    unsigned long ulHeaderMask; /* Bit Mask where 1=test, 0=ignore */

    /* Data comparator parameters */
    unsigned short uiComp1Offset; /*Offset into frame for pattern test*/
    unsigned short uiComp1Range; /* Number of bytes to test */
    unsigned char ucComp1Pattern[6]; /* Pattern to match */
    unsigned char ucComp1Mask[6]; /* Bit mask for pattern */

    unsigned short uiComp2Offset; /*Offset into frame for pattern test*/
    unsigned short uiComp2Range; /* Number of bytes to test */
    unsigned char ucComp2Pattern[6]; /* Pattern to match */
    unsigned char ucComp2Mask[6]; /* Bit mask for pattern */
} ATMTrigger; /* TRIGGER_PARAMS; */
```

**Comment**
### IT protocols

**ATM UNI**

**Description**
Set UNI timers and version

**Usage**
```c
int HTSetStructure(ATM_UNI,
0, 0, 0,
(void*)pATMUNIParams,
sizeof(ATMUNIParams),
iHub, iSlot, iPort);
```

**Related Structure**
ATMUNIParams

UNI SIGNALING PARAMS (See ATM Forum UNI 3.X specification for more info).

```c
typedef struct tagATMUNIParams
{
    unsigned long ulVer;
    unsigned long ulTmrT303;
    unsigned long ulTmrT308;
    unsigned long ulTmrT310;
    unsigned long ulTmrT313;
    unsigned long ulTmrT322;
    unsigned long ulTmrT398;
    unsigned long ulTmrT399;
    unsigned long ulTmrT309;
    unsigned long ulTmrT316;
    unsigned long ulTmrT317;
    unsigned long ulTmrTeardown;
    // } ATMUNIParams;
```  

**Comment**

### ATM - HTGetStructure

**ATM AAL5 INFO**

**Description**
Get the AAL5 layer counts

**Usage**
```c
int HTGetStructure(ATM_AAL5_INFO,
0, 0, 0,
(void*)pATMAAL5LayerInfo,
sizeof(ATMAAL5LayerInfo),
iHub, iSlot, iPort);
```

**Related Structure**
ATMAAL5LayerInfo

The ATMAAL5LayerInfo structure returns all of the counts and rates associated with the AAL5 layer.

```c
typedef struct tagATMAAL5LayerInfo
{
    unsigned long ulTimeStamp;      /* Time when the statistics were latched */
    unsigned long ulTxCell;
    unsigned long ulTxFrame;
    unsigned long ulRxCell;
    unsigned long ulRxFrame;
    unsigned long ulRxCRC32Errors;   /* ATM-2 platform only */
    unsigned long ulRxLengthErrors;   /* ATM-2 platform only */
    // } ATMAAL5LayerInfo;
```  

**Comment**
**iType1** | **ATM_CARD_CAPABILITY**  
---|---  
**Description** | Get card-specific limits  
**Usage** | `int HTGetStructure(ATM_CARD_CAPABILITY, 0, 0, 0, (void*)pATMCardCapabilities, sizeof(ATMCardCapabilities), iHub, iSlot, iPort);`  
**Related Structure** | **ATMCardCapabilities**  
Specific card capabilities are returned in this structure. This information can be used to execute code without concern for the specific card model. It can also be used, in addition to ATMCardType, to determine the specific card model revision.  
```c  
typedef struct tagATMCardCapabilities  
{  
unsigned long ulLineCellRate; /* Max rate in cells/sec */  
unsigned short uiMaxStream; /* Max number of streams */  
unsigned short uiMaxConnection; /* Max number of connections */  
unsigned short uiMaxCalls; /* Max number of active SVCs */  
unsigned short uiMaxHostTxBuffer; /* reserved */  
unsigned short uiMaxHostRxBuffer; /* reserved */  
unsigned short uiMaxLaneClients;  
unsigned short uiMaxVPITBits;  
unsigned short uiMaxVCIBits;  
unsigned short uiSupportedFeatures; /* Use one of the following defined below */  
unsigned short uiMaxRateWithoutTeardown;  
/* ushort uiReserved[3];*/  
} ATMCardCapabilities;  
```  
**Comment**
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>ATM_CARD_INFO</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Query card for firmware version</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_CARD_INFO, 0, 0, 0, (void*)pATMCardInfo, sizeof(ATMCardInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**  
ATMCardInfo

This structure will return the revision information associated with the firmware, and diagnostic and identification information associated with the hardware.

The format for the main firmware version, iMainFwVersion, is as follows:

- **Bit 15:** General release flag: Set: Released, Unset: Beta
- **Bits 0-14:** Versioning information
  - **Major release:** (ushort & 0x7FFF) /100
  - **Minor release:** (ushort & 0x7FFF) %100
  - **Build:** (ushort/1000) & 0x1F

The format for all other version/revision information is:

- **Major release:** ushort/100
- **Minor release:** ushort %100

There are no build numbers or release bits associated with these files.

typedef struct _tagATMCardInfo
{
    unsigned short   uiMainFwVersion;      /* Card firmware version*/
    unsigned short   uiSarBootFwVersion;
    unsigned short   uiSarMainFwVersion;
    unsigned short   uiPciFpgaVersion;
    unsigned short   uiGapFpgaVersion;
    unsigned short   uiBptrgFpgaVersion;
    unsigned short   uiAm29240Revision;   /* ATM-1 ONLY */
    unsigned short   uiBt8222Revision;      /* ATM-1 ONLY */
    unsigned short   uiL64363Revision;      /* ATM-1 ONLY */
    unsigned short   uiImageCheck;
    unsigned short   uiDiagFlags;
    unsigned short   uiProductCode;
} ATMCardInfo;

**Comment**

To get the major and minor release values from iMainFwVersion, you must do a bitwise AND with 7FFF. This removes the most significant bit (the General Release Flag) so that you can interpret the value.
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CARD_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get card rate (model number)</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_CARD_TYPE, 0, 0, 0, (void*)pATMCardType, sizeof(ATMCardType), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** ATMCardType

This structure contains the specified card rate. Use this number to determine the ATM card model.

```c
typedef struct tagATMCardType
{
    unsigned short uiProductCode;
    unsigned short uiReserved[15];
} ATMCardType;
```

**Comment**

Possible values for uiProductCode:
- AT9622_PRODUCT_CODE  9622
- AT9155_PRODUCT_CODE  9155
- AT9045_PRODUCT_CODE  9045
- AT9034_PRODUCT_CODE  9034
- AT9025_PRODUCT_CODE  9025
- AT9020_PRODUCT_CODE  9020
- AT9015_PRODUCT_CODE  9015
- UNSPEC_PRODUCT_CODE  9000

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CLASSICAL_IP_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>CLIP counts and ARP status</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_CLASSICAL_IP_INFO, 0, 0, 0, (void*)pATMClassicalIPInfo, sizeof(ATMClassicalIPInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** ATMClassicalIPInfo

```c
typedef struct tagATMClassicalIPInfo
{
    /* ARP server SVC state */
    unsigned char ucSvcCallState;
    unsigned char ucSvcCauseLoc;
    unsigned char ucSvcCauseCode;
    unsigned char ucReserved;

    /* Packet counters */
    unsigned short uiArpRequestPackets;
    unsigned short uiArpResponsePackets;
    unsigned short uiInarpRequestPackets;
    unsigned short uiInarpResponsePackets;
} ATMClassicalIPInfo; /* RFC1577_STATUS; */
```

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_64_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Get 64-bits status of one or many conn</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(ATM_CONN_64_INFO, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMConnection64Info, sizeof(ATMConnection64Info), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** ATMConnection64Info

ATMConnectionInfo summary structure returns the connection statistics(64-bits) associated with SmartSignaling for all of the connections in the range requested.

typedef struct tagATMConnection64Info
{| unsigned long ulIndex; unsigned long ulCount; ATMConnection64Status status[ATM_MAX_ARRAY_DIM_512]; | ATMConnection64Info; |

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_64_INFO_SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Get 64-bits summary results of sigtest</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(ATM_CONN_64_INFO_SUMMARY, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMConnection64InfoSummary, sizeof(ATMConnection64InfoSummary), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** ATMConnection64InfoSummary

typedef struct tagATMConnection64InfoSummary
{| unsigned long ulIndex; unsigned long ulCount; unsigned long ulCallsAttempted; unsigned long ulCallsEstablished; unsigned long ulCallsFailed; unsigned long ulCallsReleasedInError; unsigned long ulCallsActive; U64 ullMinRTSetupLatency; U64 ullMaxRTSetupLatency; U64 ullTotRTSetupLatency; U64 ullMinTeardownAckLatency; U64 ullMaxTeardownAckLatency; U64 ullTotTeardownAckLatency; U64 ullTestDuration; unsigned long ulFirstFailedIndex; unsigned long ulFirstActiveFailedIndex; | ATMConnection64InfoSummary; |

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get status of one or many conn</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(ATM_CONN_INFO, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMConnectionInfo, sizeof(ATMConnectionInfo), iHub, iSlot, iPort)</code></td>
</tr>
</tbody>
</table>

Related Structure: ATMConnectionInfo

ATMConnectionInfo summary structure returns the connection statistics associated with SmartSignaling for all of the connections in the range requested.

typedef struct tagATMConnectionInfo
{
    unsigned long ulIndex;
    unsigned long ulCount;
    ATMConnectionStatus status[ATM_MAX_ARRAY_DIM_512];
} ATMConnectionInfo;

Comment

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_INFO_SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get summary results of sigtest</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(ATM_CONN_INFO_SUMMARY, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMConnectionInfoSummary, sizeof(ATMConnectionInfoSummary), iHub, iSlot, iPort)</code></td>
</tr>
</tbody>
</table>

Related Structure: ATMConnectionInfoSummary

typedef struct tagATMConnectionInfoSummary
{
    unsigned long ulIndex;
    unsigned long ulCount;
    unsigned long ulCallsAttempted;
    unsigned long ulCallsEstablished;
    unsigned long ulCallsFailed;
    unsigned long ulCallsReleasedInError;
    unsigned long ulCallsActive;
    unsigned long ulMinRTSetupLatency;
    unsigned long ulMaxRTSetupLatency;
    unsigned long ulTotRTSetupLatency;
    unsigned long ulMinTeardownAckLatency;
    unsigned long ulMaxTeardownAckLatency;
    unsigned long ulTotTeardownAckLatency;
    unsigned long ulTestDuration;
    unsigned long ulFirstFailedIndex;
} ATMConnectionInfoSummary;

Comment
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_TRIGGER_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get per conn trigger counts</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_CONN_TRIGGER_INFO, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMConnTriggerInfo, sizeof(ATMConnTriggerInfo), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMConnTriggerInfo</td>
</tr>
</tbody>
</table>

ATMConnTriggerInfo records transmit and receive trigger counts on a range of virtual circuit connections.

typedef struct tagATMConnTriggerInfo
{
    unsigned short uiStartConnIndex;
    unsigned short uiConnCount;
    ATMConnTriggerStatus status[ATM_MAX_ARRAY_DIM+2]; /* index 0 & 1 are reserved */
    } ATMConnTriggerInfo;

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_DS1_E1_LINE_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get the DS1/E1 alarms and counts</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_DS1_E1_LINE_INFO, 0, 0, 0, (void*)pATMDS1E1LineInfo, sizeof(ATMDS1E1LineInfo), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMDS1E1LineInfo</td>
</tr>
</tbody>
</table>

This structure defines the statistics returned by the AT-9015 and AT-9020 cards. TBD -

typedef struct tagATMDS1E1LineInfo
{
    unsigned short uiAlarmCurrent;
    unsigned short uiAlarmHistory;
    unsigned long ulCodeViolationCount;
    unsigned long ulFrameErrorCount;
    unsigned long ulSyncErrorCount;
    unsigned long ulFebeErrorCount;
    unsigned long ulPlcpOofErrorCount;
    unsigned long ulPlcpFrameErrorCount;
    unsigned long ulPlcpBipErrorCount;
    unsigned long ulPlcpFebeErrorCount;
    unsigned long ulCodeViolationRate;
    unsigned long ulFrameErrorRate;
    unsigned long ulSyncErrorRate;
    unsigned long ulFebeErrorRate;
    unsigned long ulPlcpOofErrorRate;
    unsigned long ulPlcpFrameErrorRate;
    unsigned long ulPlcpBipErrorRate;
    unsigned long ulPlcpFebeErrorRate;
    } ATMDS1E1LineInfo;
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_DS3_E3_LINE_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Get the DS3/E3 alarms and counts</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(ATM_DS3_E3_LINE_INFO, 0, 0, 0, (void*)pATMDS3E3LineInfo, sizeof(ATMDS3E3LineInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**

ATMDS3E3LineInfo

This structure defines the statistics returned by the AT-9034 and AT-9045 cards.

typedef struct tagATMDS3E3LineInfo
{
    unsigned short  uiAlarmCurrent;
    unsigned short  uiAlarmHistory;
    unsigned long   ulCodeViolationCount;
    unsigned long   ulFrameErrorCount;
    unsigned long   ulParityErrorCount;
    unsigned long   ulCParityErrorCount;
    unsigned long   ulFebeErrorCount;
    unsigned long   ulFerfErrorCount;
    unsigned long   ulPlcpFrameErrorCount;
    unsigned long   ulPlcpBipErrorCount;
    unsigned long   ulPlcpFebeErrorCount;
    unsigned long   ulCodeViolationRate;
    unsigned long   ulFrameErrorRate;
    unsigned long   ulParityErrorRate;
    unsigned long   ulCParityErrorRate;
    unsigned long   ulFebeErrorRate;
    unsigned long   ulFerfErrorRate;
    unsigned long   ulPlcpFrameErrorRate;
    unsigned long   ulPlcpBipErrorRate;
    unsigned long   ulPlcpFebeErrorRate;
} ATMDS3E3LineInfo;

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM ELAN_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get the ELAN status</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_ELAN_INFO, &lt;index&gt;, 0, 0, (void*)pATMELANInfo, sizeof(ATMELANInfo), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMELANInfo</td>
</tr>
<tr>
<td></td>
<td>typedef struct tagATMELANInfo</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucInstance;</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucState;</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucC2Type;</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucCMTU;</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucCSName[ATM_MAX_ELAN_NAME];</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiC14LecIndex;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiCtrlDirectConnIndex;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiCtrlDistConnIndex;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiMcastSendConnIndex;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiMcastFwdConnIndex;</td>
</tr>
<tr>
<td></td>
<td>ATMAddress LesAtmAddr;</td>
</tr>
<tr>
<td></td>
<td>ATMAddress BusAtmAddr;</td>
</tr>
<tr>
<td></td>
<td>} ATMELANInfo</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Comment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_ILMI_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get the ILMI status and counts</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_ILMI_INFO, 0, 0, 0, (void*)pATMILMIInfo, sizeof(ATMILMIInfo), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMILMIInfo</td>
</tr>
<tr>
<td></td>
<td>typedef struct tagATMILMIInfo</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucState;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiColdStarts;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiGoodPackets;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiBadPackets;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiSentPackets;</td>
</tr>
<tr>
<td></td>
<td>ATMAddress RegAddr;</td>
</tr>
<tr>
<td></td>
<td>} ATMILMIInfo</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Comment</td>
</tr>
<tr>
<td>iType1</td>
<td>ATM_LAYER_INFO</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Get the ATM layer counts</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_LAYER_INFO, 0, 0, 0, (void*)pATMLayerInfo, sizeof(ATMLayerInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** ATMLayerInfo

The ATMLayerInfo structure returns all of the counts and rates associated with the ATM layer.

typedef struct tagATMLayerInfo
{
    U64 ullTxCell;
    unsigned long ulTxCellRate;
    U64 ullRxCell;
    unsigned long ulRxCellRate;
    U64 ullRxHecCorrErrors;
    unsigned long ulRxHecCorrErrorsRate;
    U64 ullRxHecUncorrErrors;
    unsigned long ulRxHecUncorrErrorsRate;
} ATMLayerInfo;

**Comment**

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_SAAL_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get the SAAL status</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_SAAL_INFO, 0, 0, 0, (void*)pATMSAALInfo, sizeof(ATMSAALInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** ATMSAALInfo

typedef struct tagATMSAALInfo
{
    unsigned char ucSscopState;
    unsigned char ucSaalState;
    unsigned long ulVtSendState;    /*ITU-T Recommendation Q.2110 p17 (pp16-22) */
    unsigned long ulVtPollSend;
    unsigned long ulVtMaxSend;
    unsigned long ulVtPollData;
    unsigned long ulVrRxState;
    unsigned long ulVrHighestExpected;
    unsigned long ulVrMaxReceive;
} ATMSAALInfo;

**Comment**
**iType1**  
**ATM_SIG_EMUL_INFO**  
**Description**  
Get the signaling emulator stats  

**Usage**  
```c  
int HTGetStructure(ATM_SIG_EMUL_INFO, 0, 0, 0,  
(void*)pATMSigEmulatorInfo,  
sizeof(ATMSigEmulatorInfo),  
iHub, iSlot, iPort) ;  
```  

**Related Structure**  
ATMSigEmulatorInfo  

typedef struct tagATMSigEmulatorInfo  
{  
  unsigned long  ulCallsHandled;  
  unsigned long  ulCallsProgressing;  
  unsigned long  ulCallsActive;  
} ATMSigEmulatorInfo;  

**Comment**

**iType1**  
**ATM_SONET_INFO**  
**Description**  
Get the SONET alarms and counts  

**Usage**  
```c  
int HTGetStructure(ATM_SONET_INFO, 0, 0, 0,  
(void*)pATMSonetLineInfo,  
sizeof(ATMSonetLineInfo),  
iHub, iSlot, iPort) ;  
```  

**Related Structure**  
ATMSonetLineInfo  

This structure defines the statistics returned by the AT-9025 and AT-9155 cards.

typedef struct tagATMSonetLineInfo  
{  
  /* Alarm conditions */  
  unsigned short  uiAlarmCurrent;  /* All alarms currently active */  
  unsigned short  uiAlarmHistory;  /* All alarms since last counter clear */  

  /* Error counts since last counter clear */  
  unsigned long  ulSectionBip8;  /* Section Bit Interleaved Parity */  
  unsigned long  ulLineBip24;  /* Line Bit Interleaved Parity */  
  unsigned long  ulLineFebe;  /* Line Far End Block Error */  
  unsigned long  ulPathBip8;  /* Path Bit Interleaved Parity */  
  unsigned long  ulPathFebe;  /* Path Far End Block Error */  

  /* Current error rates in events per second */  
  unsigned short  uiSectionBip8Rate;  
  unsigned short  uiLineBip24Rate;  
  unsigned short  uiLineFebeRate;  
  unsigned short  uiPathBip8Rate;  
  unsigned short  uiPathFebeRate;  
} ATMSonetLineInfo;  

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_STREAM_DETAIL_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get status of one or many streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_STREAM_DETAIL_INFO, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMStreamDetailedInfo, sizeof(ATMStreamDetailedInfo), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMStreamDetailedInfo</td>
</tr>
<tr>
<td></td>
<td>ATMStreamDetailedInfo allows the user to get a range of indices for detailed status.</td>
</tr>
<tr>
<td></td>
<td>typedef struct tagATMStreamDetailedInfo</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiStartIndex;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiCount;</td>
</tr>
<tr>
<td></td>
<td>ATMStreamDetailedStatus status[ATM_MAX_ARRAY_DIM];</td>
</tr>
<tr>
<td></td>
<td>} ATMStreamDetailedInfo;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_STREAM_SEARCH_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Return info for matching streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(ATM_STREAM_SEARCH_INFO, 0, 0, 0, (void*)pATMStreamSearchInfo, sizeof(ATMStreamSearchInfo), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ATMStreamSearchInfo</td>
</tr>
<tr>
<td></td>
<td>typedef struct tagATMStreamSearchInfo</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiStartIndex;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiCount;</td>
</tr>
<tr>
<td></td>
<td>unsigned short uiReturnItemId;</td>
</tr>
<tr>
<td></td>
<td>/* unsigned short uiReturnItemSize; */</td>
</tr>
<tr>
<td></td>
<td>unsigned long uiItem[1024];</td>
</tr>
<tr>
<td></td>
<td>} ATMStreamSearchInfo;</td>
</tr>
</tbody>
</table>

Comment
### iType1 ATM_TRIGGER_INFO

**Description**
ATM-1: Get trigger count and time

**Usage**
```c
int HTGetStructure(ATM_TRIGGER_INFO, 0, 0, 0, (void*)pATMTriggerInfo, sizeof(ATMTriggerInfo), iHub, iSlot, iPort);
```

<table>
<thead>
<tr>
<th>Related Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMTiggerInfo</td>
</tr>
</tbody>
</table>

The ATMTiggerInfo structure contains the time of the firing of the first trigger since the last clear counters and the number of trigger fires since the last clear counters. Used with ATM-1 only.

```c
typedef struct tagATMTriggerInfo
{
    unsigned long ulTrigger; /* Trigger event counter */
    unsigned long ulLatency; /* Latency until first trigger fired */
} ATMTiggerInfo; /* TRIGGER_STATS */
```

### iType1 ATM_VCC_INFO

**Description**
Get per VCC counts

**Usage**
```c
int HTGetStructure(ATM_VCC_INFO, <index>, <count>, 0, (void*)pATMVCCInfo, sizeof(ATMVCCInfo), iHub, iSlot, iPort);
```

<table>
<thead>
<tr>
<th>Related Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMVCCInfo</td>
</tr>
</tbody>
</table>

ATMVCCInfo records transmit and receive counts on a range of virtual circuit connections.

```c
typedef struct tagATMVCCInfo
{
    unsigned short uiIndex;
    unsigned short uiCount;
    ATMVCCIStatus status[ATM_MAX_ARRAY_DIM+2]; /* index 0 and 1 are reserved */
} ATMVCCInfo; /* VCC_STATS_HDR */
```
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_VCDB_LIST_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Get the VC DataBase info</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(ATM_VCDB_LIST_INFO,</td>
</tr>
<tr>
<td></td>
<td>&lt;index&gt;, &lt;count&gt;, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pATMVCDBInfo,</td>
</tr>
<tr>
<td></td>
<td>sizeof(ATMVCDBInfo),</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related Structure</th>
<th>ATMVCDBInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct</td>
<td>tagATMVCDBInfo</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>unsigned long ulStartEntryNum; /* Starting index */</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucEntryState; /* Of the state that we want to retrieve */</td>
</tr>
<tr>
<td></td>
<td>unsigned char ucReserved[3];</td>
</tr>
<tr>
<td></td>
<td>unsigned long ulEntryCount; /* Number of entries to retrieves */</td>
</tr>
<tr>
<td></td>
<td>ATMVCDBEntryRtvl status[ATM_MAX_ARRAY_DIM_512];</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>ATMVCDBInfo;</td>
</tr>
</tbody>
</table>

**ATM - HTSetCommand**

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CLIP_ESTABLISH_CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(ATM_CLIP_ESTABLISH_CLIENT,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No Related Structure</th>
<th>ATM_CLIP_ESTABLISH_CLIENT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CLIP_RELEASE_CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(ATM_CLIP_RELEASE_CLIENT,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No Related Structure</th>
<th>ATM_CLIP_RELEASE_CLIENT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_CONN_PARAMS_COMPLETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Reset the signaling test</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(ATM_CONN_PARAMS_COMPLETE,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No Related Structure</th>
<th>ATM_CONN_PARAMS_COMPLETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>iType1</td>
<td>ATM_CONN_PARAMS_RESET</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>Reset the signaling test conns</td>
</tr>
<tr>
<td>Usage</td>
<td><code>int HTSetCommand(ATM_CONN_PARAMS_RESET, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_FRAME_CLEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>De-allocates the &quot;Source&quot; Frame stored in memory</td>
</tr>
<tr>
<td>Usage</td>
<td><code>int HTSetCommand(ATM_FRAME_CLEAR, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_ILMI_DEREGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Deregister an ATM address</td>
</tr>
<tr>
<td>Usage</td>
<td><code>int HTSetCommand(ATM_ILMI_DEREGISTER, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_ILMI_REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Register an ATM address</td>
</tr>
<tr>
<td>Usage</td>
<td><code>int HTSetCommand(ATM_ILMI_REGISTER, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_SAAL_ESTABLISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Establish the SAAL at interface</td>
</tr>
<tr>
<td>Usage</td>
<td><code>int HTSetCommand(ATM_SAAL_ESTABLISH, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td>iType1</td>
<td>ATM_SAAL_RELEASE</td>
</tr>
<tr>
<td>--------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Release the SAAL at interface</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(ATM_SAAL_RELEASE, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_SIG_EMUL_RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Release all calls from Emulator</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(ATM_SIG_EMUL_RESET, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_START_SETUP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Start a signaling test</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(ATM_START_SETUP, &lt;index&gt;, &lt;count&gt;, 0, (void*)pATMStartCardSetupParams, iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** | ATMStartCardSetupParams

The following structure may be used to begin the signaling test on a few of the connections already established on the card. The user can use this structure to start a signaling test using a specific range of connections. Another way to start the test is to use the Start or Group Start feature which starts all configured connections.

```c
typedef struct tagATMStartCardSetupParams{
    unsigned long ulConnIndex;/* The first index for the test */
    unsigned long ulCount;     /* The subsequent indices for the test */
} ATMStartCardSetupParams;
```
<table>
<thead>
<tr>
<th>iType1</th>
<th>ATM_STOP_SETUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Stop a signaling test</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(ATM_STOP_SETUP, &lt;index&gt;, &lt;count&gt;, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pATMStopCardSetupParams,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) :</td>
</tr>
<tr>
<td>Related Structure</td>
<td>ATMSStopCardSetupParams</td>
</tr>
</tbody>
</table>

The following structure may be used to stop the signaling test on a few of the connections already established on the card. Another way to stop the test is to use the Stop or Group Stop feature which starts all configured connections. The ucStopNewCalls will prevent any more additional calls from being initiated while the test is being stopped, and the ucTeardownCalls will not only stop new calls, but tear down the existing calls as well.

typedef struct tagATMSStopCardSetupParams
{
    unsigned long ulConnIndex; /* First index to stop */
    unsigned long ulCount;     /* Subsequent indices to stop */
    unsigned char ucStopNewCalls; /* Stop new outgoing calls */
    unsigned char ucStopCellLoadGen; /* Set to 1 */
    unsigned char ucTeardownCalls; /* Stop and teardown the calls */
} ATMSStopCardSetupParams;

Comment
Chapter 4:  
10/100 MB Ethernet

This new set of Message Function parameters is for 10 MB and 100 MB Ethernet SmartCards. These same actions can also be achieved using the Original commands of the User Guide. The ETH_ parameters and structures can be used with this group of SmartCards:


The exceptions are parameters affecting MII registers which cannot be used with ST-6405 and ST-6410 SmartCards. These ETH parameters include:

ETH_WRITE_MII
ETH_FIND_MII_INFO
ETH_READ_MII_INFO

**ETH - HTSetStructure Summary**

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH_COLLISION</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHCollision</td>
<td>setup collision mode</td>
</tr>
<tr>
<td>ETH_FILL_PATTERN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UChar</td>
<td>set background pattern, using an array of unsigned chars</td>
</tr>
<tr>
<td>ETH_LATENCY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHLatency</td>
<td>setup latency test parameters</td>
</tr>
<tr>
<td>ETH_TRANSMIT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHTransmit</td>
<td>setup transmit parameters</td>
</tr>
<tr>
<td>ETH_TRIGGER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHTrigger</td>
<td>setup triggers</td>
</tr>
<tr>
<td>ETH_WRITE_MII</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHMII</td>
<td>write to a MII Address/Register</td>
</tr>
</tbody>
</table>

**ETH - HTGetStructure Summary**

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH_CARD_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHCardInfo</td>
<td>get information about a card</td>
</tr>
<tr>
<td>ETH_COUNTER_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHCounterInfo</td>
<td>get counters</td>
</tr>
<tr>
<td>ETH_ENHANCED_COUNTER_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHEnhancedCounterInfo</td>
<td>get additional counters</td>
</tr>
<tr>
<td>ETH_ENHANCED_STATUS_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHEnhancedStatusInfo</td>
<td>get status information</td>
</tr>
<tr>
<td>ETH_FIND_MII_ADDR_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHMIIInfo</td>
<td>find first MII PHY address with a legal device</td>
</tr>
<tr>
<td>ETH_LATENCY_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHLatencyInfo</td>
<td>get latency measurement report</td>
</tr>
<tr>
<td>ETH_READ_MII_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETHMIIInfo</td>
<td>read a specific MII address/reg; iType2 = PHY Address iType3 = Register</td>
</tr>
</tbody>
</table>
**ETH - HTSetCommand Summary**

<table>
<thead>
<tr>
<th>iTyp1</th>
<th>iTyp2</th>
<th>iTyp3</th>
<th>iTyp4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH_CLEAR_PORT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>clear counters</td>
</tr>
<tr>
<td>ETH_RESET_PORT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>reset the card to a default condition; iTyp2 = reset type</td>
</tr>
<tr>
<td>ETH_SELECT_RECEIVE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>select port for receive data</td>
</tr>
<tr>
<td>ETH_SELECT_TRANSMIT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>select port for transmit data iTyp2 = transmission mode</td>
</tr>
</tbody>
</table>
# ETH - HTSetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_COLLISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>setup collision mode</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(ETH_COLLISION, 0, 0, 0, (void*)pETHCollision, sizeof(ETHCollision), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

### Related Structure

**ETHCollision**

This structure sets up the collision mode for an ethernet card. The following constants have been defined:

```
uiMode: COLLISION_OFF      -- collision off
       COLLISION_LONG      -- long collision
       COLLISION_ADJ      -- adjustable collision
       CORP_A            -- collision on receive packet, Port A
       CORP_B            -- collision on receive packet, Port B
```

```c
typedef struct tagETHCollision
{
    unsigned int uiOffset; /* offset in bits where collision will take place */
    unsigned int uiDuration; /* duration in bits of the collision */
    unsigned int uiCount; /* specifies the number of consecutive collisions, up to 1024; 0 = continuous collisions */
    unsigned int uiMode; /* collision mode, see values above */
} ETHCollision;
```

### Comment

---

# ETH_FILL_PATTERN

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_FILL_PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>set background pattern, using an array of unsigned chars</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(ETH_FILL_PATTERN, 0, 0, 0, (void*)pUChar, sizeof(UChar), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

### Comment

---

SmartLib - Structure Reference
### iType1 ETH_LATENCY

**Description**
setup latency test parameters

**Usage**
```c
int HTSetStructure(ETH_LATENCY,
                  0, 0, 0,
                  (void*)pETHLatency,
                  sizeof(ETHLatency),
                  iHub, iSlot, iPort);
```

**Related Structure**
ETHLatency

This structure sets up an ethernet card for latency measurements.

```c
typedef struct tagETHLatency {
    unsigned short   uiMode;         /* latency mode:
    HT_LATENCY_OFF      -- remove the card from latency measurements
    HT_LATENCY_RX      -- set the card as a latency receiver
    HT_LATENCY_RXTX   -- set the card as a latency receiver and a
    latency transmitter
    HT_LATENCY_TX      -- set the card as a latency transmitter */
    unsigned short    uiRange;         /*   size of the pattern, in bytes */
    unsigned short   uiOffset;      /* offset of the pattern, in bits */
    unsigned char      ucPattern[12];   /* data pattern that will stop the latency
    counter */
} ETHLatency;
```

**Comment**

### iType1 ETH_TRANSMIT

**Description**
setup transmit parameters

**Usage**
```c
int HTSetStructure(ETH_TRANSMIT,
                  0, 0, 0,
                  (void*)pETHTransmit,
                  sizeof(ETHTransmit),
                  iHub, iSlot, iPort);
```

**Related Structure**
ETHTransmit

This structure sets all the basic transmit parameters. Some of the fields have constants defined for possible values.

```c
typedef struct tagETHTransmit {
    short   uiMode;         /* transmit mode:
    TX_MODE_DISABLED    -- the card is disabled
    TX_MODE_DATA       -- the card is set to transmit data
    TX_MODE_DEBUG      -- the card is set to transmit debug data
    TX_MODE_TEST       -- the card is set to transmit test data
    TX_MODE_AUTOTEST   -- the card is set to transmit autotest data
    TX_MODE_DIAG       -- the card is set to transmit diag data
} ETHTransmit;
```
typedef struct tagETHTransmit
{
    unsigned char ucTransmitMode; /* transmit mode:
    CONTINUOUS_PACKET_MODE
    SINGLE_BURST_MODE
    MULTI_BURST_MODE
    CONTINUOUS_BURST_MODE
    ECHO_MODE */
    unsigned short uiDataLength; /* number of bytes per frame, not including 4 byte CRC */

    unsigned char ucDuplexMode; /* duplex mode:
    FULLDUPLEX_MODE
    HALFDUPLEX_MODE */

    unsigned char ucSpeed; /* set card speed:
    SPEED_10MHZ
    SPEED_100MHZ
    SPEED_4MHZ
    SPEED_16MHZ */

    unsigned long ulCollisionBackoffAggressiveness; /* wait factor for backing off from multiple collisions; uses powers of 2 with this factor */

    unsigned long ulBurstCount; /* number of frames per burst */
    unsigned long ulMultiBurstCount; /* number of bursts in multi-burst */

    unsigned long ulInterFrameGap; /* length of gap between frames; 0 = random gap */
    unsigned short uiInterFrameGapScale; /* units for ulInterFrameGap: NANO_SCALE
    MICRO_SCALE
    MILLI_SCALE
    0 = bit times */

    unsigned short uiInterBurstGap; /* length of gap between bursts */

    unsigned short uiInterBurstGapScale; /* units for uiInterBurstGap, using values above; 0 = bit times */

    unsigned char ucRandomBackground; /* 1 = enable, 0 = disable */
    unsigned char ucRandomLength; /* 1 = enable, 0 = disable */

    unsigned char ucCRCErrors; /* 1 = enable, 0 = disable */
    unsigned char ucAlignErrors; /* 1 = enable, 0 = disable */
    unsigned char ucSymbolErrors; /* 1 = enable, 0 = disable */

    unsigned short uiDribbleBits; /* number of dribble bits to transmit, valid from 0 - 7 */

    unsigned char ucVFD1Mode; /* VFD1 mode: HVFD_NONE
    HVFD_RANDOM
    HVFD_INCR
    HVFD_DECR
    HVFD_STATIC */

    unsigned short uiVFD1Offset; /* offset in bits */
    short iVFD1Range; /* size of VFD1 pattern, in bytes, 6 bytes maximum; negative value signifies bit size field */

    unsigned char ucVFD1Pattern[6]; /* VFD1 pattern */
    unsigned short uiVFD1CycleCount; /* if mode is increment or decrement, specifies the number of patterns to generate before repeating */

    unsigned short uiVFD1BlockCount; /* no. of VFD1 pattern repeats before next pattern; 1=default */
}
unsigned char ucVFD2Mode; /* VFD mode, see values above */
unsigned short uiVFD2Offset; /* offset in bits */
short iVFD2Range; /* size of VFD2 pattern, in bytes, 6 bytes maximum; negative value signifies bit size field */
unsigned char ucVFD2Pattern[6]; /* VFD2 pattern */
unsigned short uiVFD2CycleCount; /* if mode is increment or decrement, specifies the number of patterns to generate before repeating */
unsigned short uiVFD2BlockCount; /* no. of VFD2 pattern repeats before next pattern: 1=default */
/* Only supported by SX-7410 */
unsigned char ucVFD3Mode; /* VFD3 mode:
HVFD_NONE
HVFD_ENABLED */
unsigned short uiVFD3Offset; /* offset of the trigger pattern, in bits */
unsigned short uiVFD3Range; /* size of the trigger pattern, in bytes, up to 2044 bytes maximum */
unsigned short uiVFD3DataCount; /* size of total VFD3 buffer */
unsigned short uiVFD3BlockCount; /* no. of VFD3 pattern repeats before next pattern: 1=default */
/* Only supported by SX-7410 */
unsigned char ucVFD3Buffer[2044]; /* VFD3 data */
unsigned char ucReserved[64];
| ETHTransmit;

Comment

**Type** ETH_TRIGGER

**Description** setup triggers

**Usage**

```c
int HTSetStructure(ETH_TRIGGER, 0, 0, 0, (void*)pETHTrigger, sizeof(ETHTrigger), iHub, iSlot, iPort);
```

**Related Structure** ETHTrigger

This structure sets up the triggers for an ethernet card.

```c
typedef struct tagETHTrigger
{
    unsigned char ucTriggerMode; /* use ETH_TRIGGER_XXX defines */
    unsigned short uiTrigger1Offset; /* offset of the trigger pattern, in bytes */
    unsigned short uiTrigger1Range; /* size of the trigger pattern, in bytes */
    unsigned char ucTrigger1Pattern[6]; /* pattern for trigger 1 */
    unsigned short uiTrigger2Offset; /* offset of the trigger pattern, in bytes */
    unsigned short uiTrigger2Range; /* size of the trigger pattern, in bytes */
    unsigned char ucTrigger2Pattern[6]; /* pattern for trigger 2 */
} ETHTrigger;
```

Comment
<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_WRITE_MII</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>write to a MII Address/Register</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(ETH_WRITE_MII, 0, 0, 0, (void*)pETHMII, sizeof(ETHMII), iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

**Related Structure** ETHMII

This structure sets an MII Address/Register for an ethernet card.

```c
typedef struct tagETHMII
{
  unsigned short uiAddress; /* specific address */
  unsigned short uiRegister; /* specific register */
  unsigned short uiValue; /* bit values to write to address/register */
} ETHMII;
```

**Comment**
### ETH - HTGetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_CARD_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>get information about a card</td>
</tr>
</tbody>
</table>

| **Usage** | int HTGetStructure(ETH_CARD_INFO, 0, 0, 0, (void*)pETHCardInfo, sizeof(ETHCardInfo), iHub, iSlot, iPort); |

#### Related Structure

**ETHCardInfo**

This structure returns information about an ethernet card.

```c
typedef struct tagETHCardInfo
{
    unsigned short   uiCardModel;      /* card model:
CM_UNKNOWN, CM_NOT_PRESENT,
CM_SE_6205, CM_SC_6305, CM_ST_6405, CM_ST_6410,
CM_SK_7205, CM_SK_7405, CM_SK_7410, CM_TR_8405,
CM_VG_7605, CM_L3_6705, CM_AT_9025, CM_AT_9155,
CM_A3_9155, CM_GX_1405, CM_AT_9015, CM_AT_9045,
CM_AT_9200, CM_AT_9020, CM_AT_9045, CM_AT_9622,
CM_L3_6710, CM_SK_7210, CM_ML_7710 */
    char               szCardModel[32];   /* card model identifier string */
    char               cPortID;            /* card type character:
'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 */
    unsigned short   uiPortType;         /* card type:
CT_ACTIVE, CT_PASSIVE, CT_FASTX, CT_TOKENRING,
CT_VG, CT_L3, CT_ATM, CT_GIGABIT, CT_ATM_SIGNALING,
CT_NOT_PRESENT */
    unsigned long      ulPortProperties;   /* card attributes bit values:
CA_SIGNALRATE_100MB -- 100 MB capable
CA_DUPLEX_FULL       -- Full Duplex capable
CA_DUPLEX_HALF       -- Half Duplex capable
CA_CONNECT_MII       -- MII connector
CA_CONNECT_TP        -- Twisted Pair connector
CA_CONNECT_BNC       -- BNC connector
CA_CONNECT_AUI       -- AUI connector
CA_CAN_ROUTE         -- Routing capable
CA_VFDORESETCOUNT    -- Resets VFDO 42 counter
CA_SIGNALRATE_4MB    -- 4 MB capable
CA_SIGNALRATE_16MB   -- 16 MB capable
CA_CAN_COLLIDE       -- Generates collisions
CA_SIGNALRATE_25MB   -- 25 MB capable
CA_SIGNALRATE_155MB  -- 155 MB capable
CA_BUILT_IPADDRESS   -- 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 */
    unsigned long      ulHWVersions[32];   /* card version information */
} ETHCardInfo;
```

**Comment**
### iType1: ETH_COUNTER_INFO

**Description**: get counters

**Usage**

```c
int HTGetStructure(ETH_COUNTER_INFO, 0, 0, 0, (void*)pETHCounterInfo, sizeof(ETHCounterInfo), iHub, iSlot, iPort);
```

**Related Structure**: ETHCounterInfo

This structure returns counter information about an ethernet card.

```c
typedef struct tagETHCounterInfo
{
    unsigned long ulRxFrames; /* number of frames received */
    unsigned long ulTxFrames; /* number of frames transmitted */
    unsigned long ulCollisions; /* number of collisions */
    unsigned long ulRxTriggers; /* number of triggers received */
    unsigned long ulRxBytes; /* number of bytes received */
    unsigned long ulCRCErrors; /* number of CRC errors received */
    unsigned long ulAlignErrors; /* number of alignment errors detected */
    unsigned long ulOversize; /* number of oversize errors detected */
    unsigned long ulUndersize; /* number of undersize errors detected */
    unsigned long ulRxFrameRate; /* number of frames received per second */
    unsigned long ulTxFrameRate; /* number of frames transmitted per second */
    unsigned long ulCRCErrorRate; /* number of CRC errors received per second */
    unsigned long ulOversizeRate; /* number of oversize errors per second */
    unsigned long ulUndersizeRate; /* number of undersize errors per second */
    unsigned long ulCollisionErrorRate; /* number of collisions per second */
    unsigned long ulAlignErrorRate; /* number of alignment errors per second */
    unsigned long ulRxTriggerRate; /* number of triggers received per second */
    unsigned long ulRxByteRate; /* number of bytes received per second */
} ETHCounterInfo;
```

**Comment**

### iType1: ETH_ENHANCED_COUNTER_INFO

**Description**: get additional counters

**Usage**

```c
int HTGetStructure(ETH_ENHANCED_COUNTER_INFO, 0, 0, 0, (void*)pETHEnhancedCounterInfo, sizeof(ETHEnhancedCounterInfo), iHub, iSlot, iPort);
```

**Related Structure**: ETHEnhancedCounterInfo

This structure returns additional counter information about an ethernet card.

```c
typedef struct tagETHEnhancedCounterInfo
{
    unsigned int uiMode;
    unsigned int uiPortType;
    unsigned long ulMask1;
    unsigned long ulMask2;
    unsigned long ulData[64];
} ETHEnhancedCounterInfo;
```

**Comment**
### DESCRIPTION

**iType1** ETH_ENHANCED_STATUS_INFO

**Description**
get status information

**Usage**

```c
int HTGetStructure(ETH_ENHANCED_STATUS_INFO, 0, 0, 0, (void*)pETHEnhancedStatusInfo, sizeof(ETHEnhancedStatusInfo), iHub, iSlot, iPort);
```

**Related Structure** ETHEnhancedStatusInfo

This structure returns status information about an ethernet card.

```c
typedef struct tagETHEnhancedStatusInfo
{
    unsigned long ulStatus; /* status bit values:
        TR_STATUS_ACCESS
        TR_STATUS_BADSTREAM
        TR_STATUS_BURST_MODE
        TR_STATUS_BEACONING
        TR_STATUS_DEVICE
        TR_STATUS_EARLY_TOKEN_RELEASE
        TR_STATUS_FULL_DUPLEX
        TR_STATUS_16MB
        TR_STATUS_RING_ALIVE
        TR_STATUS_LATENCY_STABLE
        TR_STATUS_TRANSMITTING
        GIG_STATUS_LINK
        GIG_STATUS_TX_PAUSE
        GIG_STATUS_CAPTURED_FRAMES
        GIG_STATUS_CAPTURED_STOPPED
        FAST7410_STATUS_TX_PAUSE
        L3_STATUS_6710
        VG_STATUS_MODE
        FR_STATUS_LINK_OK
        FR_STATUS_GROUP_MEMBER
        FR_STATUS_UNI_UP
        FR_STATUS_EIA_DSR
        FR_STATUS_EIA_CTS
        FR_STATUS_EIA_CDC
        FR_STATUS_EIA_TM
        FR_STATUS_EIA_DTR
        FR_STATUS_EIA_RTS
        FR_STATUS_EIA_RDL
        FR_STATUS_EIA_LDL */
} ETHEnhancedStatusInfo;
```

**Comment**

---

### DESCRIPTION

**iType1** ETH_FIND_MII_ADDR_INFO

**Description**
find first MII PHY address with a legal device

**Usage**

```c
int HTGetStructure(ETH_FIND_MII_ADDR_INFO, 0, 0, 0, (void*)pETHMIIInfo, sizeof(ETHMIIInfo), iHub, iSlot, iPort);
```

**Comment**

---

**No Related Structure**
<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_LATENCY_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>get latency measurement report</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(ETH_LATENCY_INFO, 0, 0, 0, (void*)pETHLatencyInfo, sizeof(ETHLatencyInfo), iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>ETHLatencyInfo</td>
</tr>
</tbody>
</table>

This structure returns the latency measurement for an ethernet card.

```c
typedef struct tagETHLatencyInfo {
    unsigned long ulLatency; /* latency value for the preceding latency test */
} ETHLatencyInfo;
```

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_READ_MII_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>read a specific MII address/reg; iType2 = PHY Address iType3 = Register</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(ETH_READ_MII_INFO, 0, 0, 0, (void*)pETHMIIInfo, sizeof(ETHMIIInfo), iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
</tbody>
</table>

**ETH - HTSetCommand**

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_CLEAR_PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>clear counters</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetCommand(ETH_CLEAR_PORT, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_RESET_PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>reset the card to a default condition; iType2 = reset type</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetCommand(ETH_RESET_PORT, 0, 0, 0, NULL, iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td>iType1</td>
<td>ETH_SELECT_RECEIVE</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>select port for receive data</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(ETH_SELECT_RECEIVE, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>ETH_SELECT_TRANSMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>select port for transmit data iType2 = transmission mode</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(ETH_SELECT_TRANSMIT, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5:  
100 MB Fast Ethernet

This section covers the Message Functions as related to these SmartCards:
- SX-7210
- SX-7410
- ML-7710

This chapter covers all related parameters and structures.  
All Fast Ethernet commands (iType1s) and structures work with each of these three cards with this exception:  
FST_CAPTURE_N is not used with the ML-7710 card. Instead, use L3-CAPTURE_N. 
Note that the ML-7710 card can also work with L3 commands, and certain ETH commands and structures as well.

---

**Note:** Some structures contain embedded or nested structures. In these cases, the embedded structures are included directly below the related structure.

---

### FST - HTSetStructure Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST_ALTERNATE_TX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FSTAlternateTx</td>
<td>Setup the Alt Tx stream</td>
</tr>
<tr>
<td>FST_CAPTURE_Params</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FSTCaptureParams</td>
<td>Set capture filters and controls</td>
</tr>
<tr>
<td>FST_CONTROL_Aux</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FSTControlAux</td>
<td>Set Flow Control and Preamble Len</td>
</tr>
<tr>
<td>FST_VLAN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FSTVLAN</td>
<td>Send VLAN tag information</td>
</tr>
</tbody>
</table>

### FST - HTGetStructure Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST_CAPTURE_COUNT_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FSTCaptureCountInfo</td>
<td>Get the number of captured frames</td>
</tr>
<tr>
<td>FST_CAPTURE_DATA_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FSTCaptureDataInfo</td>
<td>Get captured frame data</td>
</tr>
<tr>
<td>FST_CAPTURE_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>FSTCaptureInfo</td>
<td>Get info about captured frames</td>
</tr>
</tbody>
</table>
### FST - HTSetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>FST_ALTERNATE_TX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Setup the Alt Tx stream</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FST_ALTERNATE_TX, 0, 0, 0, (void*)pFSTAlternateTx, sizeof(FSTAlternateTx), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** | FSTAlternateTx

**Description**

Used to control the Alternate Transmit stream. This stream will be sent out after a number of normal Layer 2 frames have been sent.

```c
typedef struct tagFSTAlternateTx
{
    unsigned char ucEnabled;            /* 1 = enable, 0 = disable */
    unsigned char ucCRCErrors;         /* 1 = enable, 0 = disable */
    unsigned char ucErrorSymbol;      /* 1 = enable, 0 = disable */
    unsigned char ucDribble;           /* 1 = enable, 0 = disable */
    unsigned long ulAlternateCount;   /* no. normal frames between */
    /* each alternate frame */
    unsigned short uiDataLength;      /* length of data in bytes */
    unsigned char ucData[2048];       /* the background data buffer */
} FSTAlternateTx;
```

**Comment**

Enabling ucFilterMode will cause the fast ethernet card to only capture 64 bytes, even if uc64BytesOnly is disabled.

---

### FST - HTSetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>FST_CAPTURE_PARAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Set capture filters and controls</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FST_CAPTURE_PARAMS, 0, 0, 0, (void*)pFSTCaptureParams, sizeof(FSTCaptureParams), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure** | FSTCaptureParams

**Description**

Configure the Capture engine to capture the correct frames.

```c
typedef struct tagFSTCaptureParams
{
    unsigned char ucCRCErrors;      /* 1 = enable, 0 = disable */
    unsigned char ucOnTrigger;      /* 1 = enable, 0 = disable */
    unsigned char ucFilterMode;      /* 1 = capture filtered frames */
    /* 0 = capture all frames */
    unsigned char ucStartStopOnConditionXMode; /* 1 = stop on cond. */
    /* 0 = start on condition */
    unsigned char uc64BytesOnly;      /* 1 = enable, 0 = disable */
    unsigned char ucLast64Bytes;      /* 1 = last 64 bytes, */
    /* 0 = first 64 bytes */
    unsigned char ucCollisions;      /* 1 = enable, 0 = disable */
    unsigned char ucStartStop;      /* 1 = start capture, */
    /* 0 = stop capture */
} FSTCaptureParams;
```

**Comment**

Enabling ucFilterMode will cause the fast ethernet card to only capture 64 bytes, even if uc64BytesOnly is disabled.
### iType1 | FST_CONTROL_AUX
---|---
**Description** | Set Flow Control and Preamble Len

**Usage** | int HTSetStructure(FST_CONTROL_AUX, 0, 0, 0, (void*)pFSTControlAux, sizeof(FSTControlAux), iHub, iSlot, iPort);

**Related Structure** | FSTControlAux

Auxiliary control parameters to enable flow control pause frames and to set the preamble length.

```c
typedef struct tagFSTControlAux {
    unsigned char ucFlowControlPause; /* 1 = enable, 0 = disable */
    unsigned char ucPreambleLen;       /* preamble length in bits */
    /* (valid: 16, 32, 48, 64) */
    /* ML-7710 has a fixed preamble */
    /* so for this card, set to 0 */
} FSTControlAux;
```

### Comment

### iType1 | FST_VLAN
---|---
**Description** | Send VLAN tag information

**Usage** | int HTSetStructure(FST_VLAN, 0, 0, 0, (void*)pFSTVLAN, sizeof(FSTVLAN), iHub, iSlot, iPort);

**Related Structure** | FSTVLAN

Enables and configures all Virtual LAN header values.

```c
typedef struct tagFSTVLAN {
    unsigned char  ucVLANEnable;       /* 1 = enable, 0 = disable    */
    unsigned short uiTPID;     /* VLAN type (2 bytes)                */
    unsigned char  ucPRI;      /* user priority 0-7 (3 bits)         */
    /* use VLAN_PRI_XXX definitions */
    unsigned char  ucCFI;      /* 1=RIF present, 0=RIF absent (1 bit)*/
    /* use VLAN_CFI_XXX definitions */
    unsigned short uiVID;      /* VLAN ID (12 bits)                  */
} FSTVLAN;
```

### Comment
### FST - HTGetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>FST_CAPTURE_COUNT_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Get the number of captured frames</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(FST_CAPTURE_COUNT_INFO, 0, 0, 0, (void*)pFSTCaptureCountInfo,</td>
</tr>
<tr>
<td></td>
<td>sizeof(FSTCaptureCountInfo), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Related Structure</td>
<td>FSTCaptureCountInfo</td>
</tr>
</tbody>
</table>

Provides a structure in which to get how many frames have been captured on this SmartCard.

```c
typedef struct tagFSTCaptureCountInfo
{
    unsigned long ulCaptureCount; /* number of captured packets */
} FSTCaptureCountInfo;
```

### Comment

### iType1

<table>
<thead>
<tr>
<th>FST_CAPTURE_DATA_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Usage</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Related Structure</td>
</tr>
</tbody>
</table>

Used to retrieve the data of the frame which has been captured.

```c
typedef struct tagFSTCaptureDataInfo
{
    unsigned long ulFrameNum; /* frame number (input) */
    unsigned char ucData[2048];
} FSTCaptureDataInfo;
```

### Comment

### iType1

<table>
<thead>
<tr>
<th>FST_CAPTURE_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Usage</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Related Structure</td>
</tr>
</tbody>
</table>

```c
typedef struct tagFSTCaptureInfo
{
    FSTCaptureFrameInfo FrameInfo[FST_MAX_CAPTURE_FRAMES];
} FSTCaptureInfo;
```

### Comment
Chapter 6:
Gigabit Ethernet

**GIG - HTSetStructure Summary**

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIG STRUCT ALT TX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGAltTransmit</td>
<td>setup alternate transmit</td>
</tr>
<tr>
<td>GIG STRUCT AUTO FIBER NEGOTIATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGAutoFiberNegotiate</td>
<td>setup AutoFiber Negotiation</td>
</tr>
<tr>
<td>GIG STRUCT_BG1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UChar</td>
<td>set background pattern for alternate frame, using an array of unsigned chars</td>
</tr>
<tr>
<td>GIG STRUCT_BG2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UChar</td>
<td>background pattern for periodic frame, using an array of unsigned chars</td>
</tr>
<tr>
<td>GIG STRUCT_CAPTURE_SETUP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGCaptureSetup</td>
<td>setup capture</td>
</tr>
<tr>
<td>GIG STRUCT_FILL_PATTERN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UChar</td>
<td>set main background pattern,</td>
</tr>
<tr>
<td>GIG STRUCT_TRIGGER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGTrigger</td>
<td>setup triggers</td>
</tr>
<tr>
<td>GIG STRUCT_TX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGTransmit</td>
<td>setup transmit</td>
</tr>
<tr>
<td>GIG STRUCT_VFD3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UChar</td>
<td>set VFD3 data,</td>
</tr>
</tbody>
</table>

**GIG - HTGetStructure Summary**

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIG STRUCT CAP COUNT INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGCaptureCountInfo</td>
<td>get number of frames captured</td>
</tr>
<tr>
<td>GIG STRUCT CAP DATA INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGCaptureDataInfo</td>
<td>get a frame’s captured data</td>
</tr>
<tr>
<td>GIG STRUCT CAP INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>GIGCaptureInfo</td>
<td>get info about captured frames</td>
</tr>
<tr>
<td>GIG STRUCT CARD INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGCardInfo</td>
<td>get card information</td>
</tr>
<tr>
<td>GIG STRUCT COUNTER INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGCounterInfo</td>
<td>get counter information</td>
</tr>
<tr>
<td>GIG STRUCT IMAGE VERSIONS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGVersions</td>
<td></td>
</tr>
<tr>
<td>GIG STRUCT RATE INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GIGRateInfo</td>
<td>get rate information</td>
</tr>
</tbody>
</table>
**GIG - HTSetStructure**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>IType1</td>
<td>GIG_STRUC_ALT_TX</td>
<td>int HTSetStructure(GIG_STRUC_ALT_TX, 0, 0, 0, (void*)pGIGAltTransmit, sizeof(GIGAltTransmit), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**

GIGAltTransmit

This structure sets some additional transmit parameters, specifically to enable the alternate and periodic frames and to enable PAUSE frame recognition. Note that the periodic frame will begin transmitting as soon as this structure is sent with ucEnableBG2 set to 1. It will continue transmitting with the periodic rate set by the GIGTransmit structure (ulBG2Frequency field) until the GIGAltTransmit structure is sent again with ucEnableBG2 set to 0.

```c
typedef struct tagGIGAltTransmit {
    unsigned char ucEnableSS1;     /* reserved: 0 = default */
    unsigned char ucEnableSS2;     /* reserved: 0 = default */
    unsigned char ucEnableBG1;     /* enable alternate frame (0 = default) */
    unsigned char ucEnableBG2;     /* enable periodic frame (0 = default) */
    unsigned char ucEnableHoldoff; /* enable PAUSE frame recognition */
    unsigned char ucReserved[3];   /* reserved: 0 = default */
} GIGAltTransmit;
```
**iType1** | **GIG_STRUC_AUTO_FIBER_NEGOTIATE**  
--- | ---  
**Description** | setup AutoFiber Negotiation  
**Usage** | ```c  
int HTSetStructure(GIG_STRUC_AUTO_FIBER_NEGOTIATE,  
0, 0, 0,  
(void*)pGIGAutoFiberNegotiate,  
sizeof(GIGAutoFiberNegotiate),  
iHub, iSlot, iPort);  
```  
**Related Structure** | GIGAutoFiberNegotiate  
**Comment** |

```c  
typedef struct tagGIGAutoFiberNegotiate  
{  
unsigned char ucMode; /* 1 = enable auto-negotiation,  
0 = disable (0 = default) */  
unsigned char ucRestart; /* 1 = restart auto-negotiation,  
0 = do not restart (0 = default) */  
unsigned short uiLinkConfiguration; /* reserved  
-- use the uiLinkConfiguration field  
in GIGTransmit (0 = default) */  
unsigned char ucEnableCCode; /* 1 = Use C Code (uiLinkConfiguration)  
for negotiation, 0 = or not (0 = default) */  
unsigned char ucReserved[7]; /* reserved -- not currently in use  
(0 = default) */  
} GIGAutoFiberNegotiate;  
```
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>GIG_STRUC_BG1</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>set background pattern for alternate frame, using an array of unsigned chars</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(GIG_STRUC_BG1, 0, 0, 0, (void*)pUChar, sizeof(UChar), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>GIG_STRUC_BG2</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>background pattern for periodic frame, using an array of unsigned chars</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(GIG_STRUC_BG2, 0, 0, 0, (void*)pUChar, sizeof(UChar), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Comment**
**Description**

setup capture

**Usage**

```c
int HTSetStructure(GIG_STRUC_CAPTURE_SETUP, 0, 0, 0, (void*)pGIGCaptureSetup, sizeof(GIGCaptureSetup), iHub, iSlot, iPort);
```

**Related Structure**

GIGCaptureSetup

This structure sets the capture parameters. If ucFilterMode is enabled, the card will only capture frames with CRC errors, Rx triggers, Tx triggers, or RC errors, as determined by the ucCRCErrors, ucRxTriggers, ucTxTriggers, and ucRCErrors fields, respectively. If ucStartStopOnConditionMode is enabled, the card will START capturing upon receiving a frame meeting the conditions determined by the first four parameters (ucCRCErrors, etc). If it is disabled, the card will STOP capturing upon receiving a frame meeting those conditions. If ucFilterMode is disabled, the card will capture all frames (up to a maximum of 2048). Use the ucStartStop field to start or stop capturing frames. Capture must be stopped before examining any captured data.

```c
typedef struct tagGIGCaptureSetup
{
    unsigned char ucCRCErrors; /* 1 = enable, 0 = disable: 0=default*/
    unsigned char ucRxTrigger; /* 1 = enable, 0 = disable: 0=default*/
    unsigned char ucTxTrigger; /* 1 = enable, 0 = disable: 0=default*/
    unsigned char ucRCErrors;   /* 1 = enable, 0 = disable: 0=default*/
    unsigned char ucFilterMode; /* 1 = capture filtered frames, 0 = capture all frames: 0=default */
    unsigned char ucLast64Bytes; /* 1=last 64 bytes, 0 = first 64 bytes (0=default) */
    unsigned char ucStartStop;  /* 1=start capture, 0 = stop capture (0=default) */
} GIGCaptureSetup;
```

**Comment**

The gigabit card hardware imposes a limitation on capturing trigger frames. When capturing trigger frames only (ucFilterMode = 1, and ucRxTrigger = 1 or ucTxTrigger = 1) the interframe gap must be at least 128ns. If the gap is set to the gigabit card minimum of 96ns, the card will capture the frame following the trigger frame instead. Also, enabling ucFilterMode will cause the gigabit card to only capture 64 bytes, even if uc64BytesOnly is disabled.
<table>
<thead>
<tr>
<th>iType1</th>
<th>GIG_STRUC_FILL_PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>set main background pattern,</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(GIG_STRUC_FILL_PATTERN, 0, 0, 0, (void*)pUChar, sizeof(UChar), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>GIG_STRUC_TRIGGER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>setup triggers</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(GIG_STRUC_TRIGGER, 0, 0, 0, (void*)pGIGTrigger, sizeof(GIGTrigger), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>GIGTrigger</td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td>This structure sets up the triggers for a gigabit card. This structure is similar to the HTTrigger function. The ucTriggerMode field can be set to specify what combination of triggers must occur to create a trigger event.</td>
</tr>
</tbody>
</table>

```c
typedef struct tagGIGTrigger
{
    unsigned char ucTrigger1Mode;    /* reserved - not currently in use
    (0 = default) */
    unsigned char ucTrigger1Range;   /* range of trigger 1 (in bytes)
    (0 = default) */
    unsigned short uiTrigger1Offset; /* offset of trigger 1 (in bits)
    (0 = default) */
    unsigned char ucTrigger1Data[8]; /* trigger 1 data (0 = default) */
    unsigned char ucTrigger1Mask[8]; /* trigger 1 mask
    (0xFF = defaults) */
    unsigned char ucTrigger2Mode;    /* reserved - not currently in use
    (0 = default) */
    unsigned char ucTrigger2Range;   /* range of trigger 2 (in bytes)
    (0 = default) */
    unsigned short uiTrigger2Offset; /* offset of trigger 2 (in bits)
    (0 = default) */
    unsigned char ucTrigger2Data[8]; /* trigger 2 data (0 = default) */
    unsigned char ucTrigger2Mask[8]; /* trigger 2 mask
    (0xFF = default) */
    unsigned char ucTriggerMode;     /* use trigger combination defines
    (GIG_TRIGGER1_ONLY = default) */
    unsigned char ucReserved;        /* reserved - not currently in use
    (0 = default) */
} GIGTrigger;
```
<table>
<thead>
<tr>
<th>iType1</th>
<th>GIG STRUCT TX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>setup transmit</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetStructure(GIG_STRUCT_TX, 0, 0, 0, (void*)pGIGTransmit, sizeof(GIGTransmit), iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>GIGTransmit</td>
</tr>
</tbody>
</table>

This structure sets all the basic transmit parameters for the gigabit card, including the VFDs. References to "Main" frame refer to the standard frame (as determined by the transmit parameters); references to the "BG1" frame refer to the alternate frame; and references to the "BG2" frame refer to the periodic frame. Background data must be set using GIG_STRUCT_FILL_PATTERN, GIG_STRUCT_BG1, and GIG_STRUCT_BG2, all with arrays of unsigned chars, for the main frame, alternate frame, and periodic frame, respectively. Also, while the VFD3 parameters are set in the GIGTransmit structure, the VFD3 data itself is set using GIG_STRUCT_VFD3 with an array of unsigned chars.

```c
typedef struct tagGIGTransmit
{
    unsigned short uiMainLength;    /* length of main packet (bytes)
                                   (60 = default) */
    unsigned char  ucPreambleByteLength;   /* preamble length (bytes)
                                             (8 = default) */
    unsigned char  ucFramesPerCarrier;     /* reserved, forced to 1
                                              (1 = default) */
    unsigned long  ulGap;                  /* interframe gap (ns)
                                               (96 = default) */
    unsigned char  ucMainRandomBackground; /* 1 = enable, 0 = disable
                                              (0 = default) */
    unsigned char  ucBG1RandomBackground;  /* 1 = enable, 0 = disable
                                            (0 = default) */
    unsigned char  ucBG2RandomBackground;  /* 1 = enable, 0 = disable
                                             (0 = default) */
    unsigned char  ucSS1RandomBackground;  /* 1 = enable, 0 = disable
                                             (0 = default) */
    unsigned char  ucSS2RandomBackground;  /* 1 = enable, 0 = disable
                                             (0 = default) */
    unsigned char  ucMainCRCError; /*1 = enable, 0 = disable:0=default*/
    unsigned char  ucBG1CRCError; /*1 = enable, 0 = disable:0=default*/
    unsigned char  ucBG2CRCError; /*1 = enable, 0 = disable:0=default*/
    unsigned char  ucSS1CRCError; /*1 = enable, 0 = disable:0=default*/
    unsigned char  ucSS2CRCError; /*1 = enable, 0 = disable:0=default*/
    unsigned char  ucJabberCount;  /* for each unit, add 8K bytes to simulate jabber (0 = default) */
    unsigned char  ucLoopback;     /* 1 = enable, 0 = disable
                                             (0 = default) */
} GIGTransmit;
```
unsigned long ulBG1Frequency; /* no. of main frames between each alternate frame (0 = default) */
unsigned long ulBG2Frequency; /* gap between periodic frames, in units of 32ns (0 = default) */

unsigned short uiBG1Length;    /* length of alternate frame (bytes) (0 = default) */
unsigned short uiBG2Length;    /* length of periodic frame (bytes) (0 = default) */
unsigned short uiSS1Length;    /* reserved -- not currently in use (0 = default) */
unsigned short uiSS2Length;    /* reserved -- not currently in use (0 = default) */

unsigned short ulLinkConfiguration; /* use GIG_AFN_XXX values below (0x20 = default) */

unsigned short ulVFD1Offset;    /* bit offset of VFD1: 0=default */
short iVFD1Range;    /* range of VFD1 (no. of bytes) (0 = default) */

unsigned char ucVFD1Mode;     /* see GIG_VFD_XXX definitions (GIG_VFD_OFF = default) */
unsigned long ulVFD1CycleCount; /* no. of frames before resetting to initial VFD1 pattern: 0=default*/
unsigned char ucVFD1Data[8];  /* initial VFD1 data (0 = default) */
unsigned short uiVFD2Offset; /* bit offset of VFD2: 0 = default */
short iVFD2Range; /* range of VFD2 (number of bytes) (0 = default) */
unsigned char ucVFD2Mode; /* see GIG_VFD_XXX definitions (GIG_VFD_OFF = default) */
unsigned long ulVFD2CycleCount; /* no. of frames before resetting to initial VFD2 pattern: 0=default */
unsigned char ucVFD2Data[8]; /* initial VFD2 data: 0=default */

unsigned short uiVFD3Offset; /* bit offset of VFD3: 0 = default */
unsigned short uiVFD3Range; /* range of VFD3 (number of bytes of VFD3 data per frame) (0 = default) */
unsigned long ulVFD3Count; /* no. of frames worth of VFD3 data (0 = default) */
unsigned char ucVFD3Mode; /* see GIG_VFD3_XXX definitions (GIG_VFD3_OFF = default) */

unsigned char ucMainBG1Mode; /* reserved -- not currently in use (0 = default) */
unsigned long ulBurstCount; /* number of frames per burst; when ucTransmitMode is set to GIG.Continuous_MODE, ulBurstCount must be set to a value greater than 0 (1 = default) */
unsigned long ulMultiBurstCount; /* no. of bursts per multi-burst (1 = default) */
unsigned long ulInterBurstGap; /* gap between multi-bursts (nanoseconds) (0 = default) */
unsigned char ucTransmitMode; /* see GIG_xxx_MODE definitions (GIG.Continuous_MODE = default) */
unsigned char ucEchoMode; /* 1 = enable, 0 = disable (0 = default) */
unsigned char ucPeriodicGap; /* gap between main frame and periodic frame, in increments of 32ns; valid values from 0 to 7 (0 to 224ns), but less than 3 defaults to 3 (96ns) (0 = default) */
unsigned char ucCountRcverrOrOvrsz; /* 0 = count oversized pkt errs 1 = count receive errors (0 = default) */
unsigned char ucGapByBitTimesOrByRate; /* 0 = set gap by bit times, 1 = set gap by Tx rate (0 = default) */
unsigned char ucRandomLengthEnable; /* 1 = enable random pkt length 0 = disable (0 = default) */
unsigned short uiVFD1BlockCount; /* no. of VFD1 pattern repeats before next pattern: 1=default */
unsigned short uiVFD2BlockCount; /* no. of VFD2 pattern repeats before next pattern: 1=default */
unsigned short uiVFD3BlockCount; /* no. of VFD3 pattern repeats before next pattern: 1=default */
unsigned char ucExtra[2]; /* reserved -- not currently in use (0 = default) */

Comment

The ulBurstCount field should be set to a value greater than 0 when ucTransmit mode is set to GIG.Continuous_MODE. Otherwise, the ulGap value will not take effect, and frames will be transmitted with a minimum gap. Also, to ensure correct card behavior, the ucFramesPerCarrier field should be set to 1.

<table>
<thead>
<tr>
<th>Type1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GIG_STRUCT_VFD3</td>
</tr>
</tbody>
</table>

set VFD3 data.
### GIG - HTSetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>GIG_STRUC_VFD3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>set VFD3 structure</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(GIG_STRUC_VFD3, 0, 0, 0, (void*)pUChar, sizeof(UChar), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**No Related Structure**

### GIG - HTGetStructure

- **GIG - HTGetStructure**
  - **iType1** | GIG_STRUC_CAP_COUNT_INFO |
  - **Description** | get number of frames captured |
  - **Usage** | int HTGetStructure(GIG_STRUC_CAP_COUNT_INFO, 0, 0, 0, (void*)pGIGCaptureCountInfo, sizeof(GIGCaptureCountInfo), iHub, iSlot, iPort); |
  - **Related Structure** | GIGCaptureCountInfo |

This structure returns the total number of frames captured.

```c
typedef struct tagGIGCaptureCountInfo
{
    unsigned long ulCount; /* no. of captured packets (0 to 2048) */
} GIGCaptureCountInfo;
```

### GIG - HTGetStructure

- **GIG - HTGetStructure**
  - **iType1** | GIG_STRUC_CAP_DATA_INFO |
  - **Description** | get a frame's captured data |
  - **Usage** | int HTGetStructure(GIG_STRUC_CAP_DATA_INFO, 0, 0, 0, (void*)pGIGCaptureDataInfo, sizeof(GIGCaptureDataInfo), iHub, iSlot, iPort); |
  - **Related Structure** | GIGCaptureDataInfo |

This structure returns the data of a captured frame. Only one frame's data is retrieved by each call.

Set the frame index relative to all captured frames in the ulFrame field BEFORE calling HTGetStructure. Then the data for that frame will be returned in ucData.

```c
typedef struct tagGIGCaptureDataInfo
{
    unsigned long ulFrame;       /* frame number (input) */
    unsigned char  ucData[2048]; /* captured frame data */
} GIGCaptureDataInfo;
```
## Description
Get information about captured frames.

## Usage
```c
int HTGetStructure(GIG_STRUC_CAP_INFO,  
                  <index>, <count>, 0,  
                  (void*)pGIGCaptureInfo,  
                  sizeof(GIGCaptureInfo),  
                  iHub, iSlot, iPort);
```

## Related Structure
**GIGCaptureInfo**

This structure receives information about captured frames. The GIGCaptureInfo structure consists of an array of GIGCaptureFrameInfo structures. This allows you to receive information about GIG_MAX_CAPTURE_FRAMES (currently 96) frames.

To use this structure, use HTGetStructure with an iType1 of GIG_STRUC_CAP_INFO and a pData of a pointer to a GIGCaptureInfo structure. Also, iType2 must contain the index of the first captured frame to get information about, and iType3 must contain the index of the frame after the last captured frame to get information about. The structure will then return with the FrameInfo array filled with structures describing all the captured frames within the boundaries specified by iType2 and iType3. The FrameInfo structures will give the frame index (relative to all captured frames), a timestamp, a status indicator, and the length of the captured frame. The status field (uiStatus) indicates the status of the frame based on certain bits being set, as specified by the GIG_CAP_XXX definitions. If you perform a bit-wise "AND" (&) on uiStatus with the status characteristic bit to check, you can determine whether or not that characteristic applies to that frame.

```c
typedef struct tagGIGCaptureInfo  
{  
    GIGCaptureFrameInfo FrameInfo[GIG_MAX_CAPTURE_FRAMES];  
} GIGCaptureInfo;
```

## Comment

### iType1
**GIG_STRUC_CARD_INFO**

Get card information.

## Usage
```c
int HTGetStructure(GIG_STRUC_CARD_INFO,  
                   0, 0, 0,  
                   (void*)pGIGCardInfo,  
                   sizeof(GIGCardInfo),  
                   iHub, iSlot, iPort);
```

## Related Structure
**GIGCardInfo**

This structure returns information about the current state of the gigabit card.

```c
typedef struct tagGIGCardInfo  
{  
    unsigned short uiLinkConfiguration; /* current link configuration */  
    unsigned int ulLinkStateChanges; /* no. of link state changes */  
} GIGCardInfo;
```

## Comment
**IType1** | **GIG_STRUCT_COUNTER_INFO**  
---|---  
**Description** | get counter information  
**Usage** | int HTGetStructure(GIG_STRUCT_COUNTER_INFO, 0, 0, 0, (void*)pGIGCounterInfo, sizeof(GIGCounterInfo), iHub, iSlot, iPort) ;  
**Related Structure** | GIGCounterInfo  
* This structure returns counter information about the gigabit card.  
*  
```c  
typedef struct tagGIGCounterInfo  
{  
    U64 ullTxFrames;   /* no. of transmitted frames */  
    U64 ullTxBytes;    /* no. of transmitted bytes */  
    U64 ullTxTriggers; /* no. of transmit triggers */  
    unsigned long ulTxLatency; /* Tx latency counter (50ns units) */  
    U64 ullRxFrames;   /* no. of received frames */  
    U64 ullRxBytes;    /* no. of received bytes */  
    U64 ullRxTriggers; /* no. of receive triggers */  
    unsigned long ulRxLatency; /* Rx latency counter (50ns units) */  
    U64 ullCMCErrors; /* no. of frames with CRC errors */  
    U64 ullOverSize;   /* no. of oversized frames */  
    U64 ullUnderSize;  /* no. of undersized frames */  
} GIGCounterInfo;  ```  
**Comment**

**IType1** | **GIG_STRUCT_IMAGE_VERSIONS**  
---|---  
**Description** |  
**Usage** | int HTGetStructure(GIG_STRUCT_IMAGE_VERSIONS, 0, 0, 0, (void*)pGIGVersions, sizeof(GIGVersions), iHub, iSlot, iPort) ;  
**Related Structure** | GIGVersions  
```c  
typedef struct tagGIGVersions  
{  
    unsigned short uiFirmwareVersion;  
    unsigned short uiTransmitDataVersion;  
    unsigned short uiTransmitControlVersion;  
    unsigned short uiTransmitLowlevelVersion;  
    unsigned short uiReceiveDataVersion;  
    unsigned short uiReceiveControlVersion;  
    unsigned short uiReceiveLowlevelVersion;  
    unsigned short uiBackplaneControlVersion;  
    unsigned short uiLinkControlVersion;  
    unsigned char ucFirmwareCheck;  
    unsigned char ucTransmitControlCheck;  
    unsigned char ucTransmitLowlevelCheck;  
    unsigned char ucReceiveDataCheck;  
    unsigned char ucReceiveControlCheck;  
    unsigned char ucReceiveLowlevelCheck;  
    unsigned char ucBackplaneControlCheck;  
    unsigned char ucLinkControlCheck;  
    unsigned char ucBootVersion;  
    unsigned char ucReserved[16];  
} GIGVersions;  ```  
**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>GIG_STRUCT_RATE_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>get rate information</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(GIG_STRUCT_RATE_INFO, 0, 0, 0, (void*)pGIGRateInfo, sizeof(GIGRateInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**

GIGRateInfo

This structure returns rate counter information about the gigabit card.

```c
typedef struct tagGIGRateInfo
{
    unsigned long ulTxFrames;     /* transmitted frames per second */
    unsigned long ulTxBytes;      /* transmitted bytes per second */
    unsigned long ulTxTriggers;   /* transmit triggers per second */
    unsigned long ulRxFrames;     /* received frames per second */
    unsigned long ulRxBytes;      /* received bytes per second */
    unsigned long ulRxTriggers;   /* receive triggers per second */
    unsigned long ulCRCErrors;    /* frames with CRC errors per second */
    unsigned long ulOverSize;     /* oversized frames per second */
    unsigned long ulUnderSize;    /* undersized frames per second */
} GIGRateInfo;
```
Chapter 7:
L3 - Ethernet with SmartMetrics

This section covers the Message Functions as related to these SmartCards:
- L3-6705
- L3-6710
- ML-7710

All L3 commands and structures work with each of these cards. This chapter covers all related parameters and structures.

Note that the ML-7710 card can also work with certain FST and ETH commands and structures as well.

For additional information about L3 parameters, see Test Results with SmartMetrics Histograms on page 14.

---

Note: Some structures contain embedded or nested structures. In these cases, the embedded structures are included directly below the related structure.

---

**L3 - HTSetStructure Summary**

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3_DEFINE_IP_STREAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>StreamIP</td>
<td>Create a list with one or more IP compliant Tx Streams</td>
</tr>
<tr>
<td>L3_DEFINE_IPX_STREAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>StreamIPX</td>
<td>Create a list with one or more IPX compliant Tx Streams</td>
</tr>
<tr>
<td>L3_DEFINE_MULTI_IP_STREAM</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>StreamIP</td>
<td>Append similar copies of an IP Stream</td>
</tr>
<tr>
<td>L3_DEFINE_MULTI_IPX_STREAM</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>StreamIPX</td>
<td>Append similar copies of an IPX Stream</td>
</tr>
<tr>
<td>L3_DEFINE_MULTI_SMARTBITS_STREAM</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>StreamSmartBits</td>
<td>Append similar copies of a SmartBits Stream</td>
</tr>
<tr>
<td>L3_DEFINE_MULTI_UDP_STREAM</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>StreamUDP</td>
<td>Append similar copies of a UDP Stream</td>
</tr>
<tr>
<td>L3_DEFINE_SMARTBITS_STREAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>StreamSmartBits</td>
<td>Create a list with one or more customizable Streams.</td>
</tr>
<tr>
<td>L3_DEFINE_UDP_STREAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>StreamUDP</td>
<td>Create a list with one or more UDP compliant Tx Streams</td>
</tr>
<tr>
<td>L3_MOD_IP_STREAM</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>StreamIP</td>
<td>Overwrite one IP Stream in the list at a given index</td>
</tr>
<tr>
<td>L3_MOD_IPX_STREAM</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>StreamIPX</td>
<td>Overwrite one IPX Stream in the list at a given index</td>
</tr>
<tr>
<td>L3_MOD_SMARTBITS_STREAM</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>StreamSmartBits</td>
<td>Overwrite one SmartBits Stream in the list at a given index</td>
</tr>
<tr>
<td>L3_MOD_STREAMS_ARRAY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3ModifyStreamArray</td>
<td>Modify a field in a block of streams by overlaying an element from an array</td>
</tr>
<tr>
<td>L3_MOD_STREAMS_DELTA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3ModifyStreamDelta</td>
<td>Increment a field in a block of streams by incrementing a base value</td>
</tr>
</tbody>
</table>
## L3 - HTGetStructure Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3_ARP_TIMES_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ULong</td>
<td>Retrieve ARP Times for each Stream in the list</td>
</tr>
<tr>
<td>L3_CAPTURE_COUNT_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3CaptureCountInfo</td>
<td>Get the count of captured frames</td>
</tr>
<tr>
<td>L3_CAPTURE_PACKET_DATA_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>Layer3CaptureData</td>
<td>Get the captured frame</td>
</tr>
<tr>
<td>L3_DEFINED_STREAM_COUNT_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ULong</td>
<td>retrieve the count of Streams in the list</td>
</tr>
<tr>
<td>L3_HIST_ACTIVE_TEST_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3HistActiveTest</td>
<td>Get the number of histogram records, and active histogram</td>
</tr>
<tr>
<td>L3_HIST_LATENCY_DISTRIBUTION_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>Layer3StreamDistributionInfo</td>
<td>Get Latency Distribution histogram results</td>
</tr>
<tr>
<td>L3_HIST_RAW_TAGS_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>Layer3HistTagInfo</td>
<td>Get Raw Tags histogram records</td>
</tr>
<tr>
<td>L3_HIST_SEQUENCE_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>Layer3SequenceInfo</td>
<td>Get Sequence Tracking histogram results</td>
</tr>
<tr>
<td>L3_HIST_V2_LATENCY_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>Layer3LongLatencyInfo</td>
<td>Get Latency over Time histogram results</td>
</tr>
<tr>
<td>L3_HIST_V2_LATENCY_PER_STREAM_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>Layer3StreamLongLatencyInfo</td>
<td>Get combination histogram results - Latency Dist, Lat. Per Stream, Sequence.</td>
</tr>
<tr>
<td>L3_STREAM_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>StreamSmartBits</td>
<td>retrieve Streams setup information in the Streams list</td>
</tr>
<tr>
<td>L3_TX_ADDRESS_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3Address</td>
<td>SmartCard device address information</td>
</tr>
</tbody>
</table>

## L3 - HTSetCommand Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3_CAPTURE_ALL_TYPE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Capture all frames received</td>
</tr>
<tr>
<td>L3_CAPTURE_BAD_TYPE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Capture errors frames only</td>
</tr>
<tr>
<td>L3_CAPTURE_OFF_TYPE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Turn Capture off</td>
</tr>
<tr>
<td>L3_CAPTURE_TRIGGERS_TYPE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Capture Trigger frames only</td>
</tr>
<tr>
<td>L3_HIST_LATENCY_DISTRIBUTION</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3HistDistribution</td>
<td>Get Latency Distribution histogram results</td>
</tr>
<tr>
<td>L3_HIST_RAW_TAGS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Get Raw Tags histogram records</td>
</tr>
<tr>
<td>L3_HIST_SEQUENCE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Get Sequence Tracking histogram results</td>
</tr>
<tr>
<td>L3_HIST_START</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Clear Histogram records/Histogram remains in Receive state.</td>
</tr>
<tr>
<td>L3_HIST_V2_LATENCY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3HistLatency</td>
<td>Get Latency over Time histogram results</td>
</tr>
<tr>
<td>L3_HIST_V2_LATENCY_PER_STREAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3V2HistDistribution</td>
<td>Get combination histogram results - Latency Dist, Lat. Per Stream, Sequence.</td>
</tr>
<tr>
<td>L3_START_ARPS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Begin ARP exchange on all defined Streams</td>
</tr>
</tbody>
</table>
**L3 - HTSetStructure**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3_DEFINE_IP_STREAM</td>
<td>Create a list with one or more IP compliant Tx Streams</td>
</tr>
</tbody>
</table>

**Usage**

```c
int HTSetStructure(L3_DEFINE_IP_STREAM, 0, 0, 0, (void*)pStreamIP, sizeof(StreamIP), iHub, iSlot, iPort);
```

**Related Structure**

StreamIP

This structure is used to define IP compliant Streams.

The IP checksum is automatically calculated using the supplied header fields and inserted into the IP header.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamIP
{
    unsigned char ucActive;         /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType;   /* use STREAM_PROTOCOL_IP */
    unsigned char ucRandomLength;   /* Reserved */
    unsigned char ucRandomData;     /* Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength;   /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset;    /* in bits */
    unsigned char ucVFD1Range;      /* in bits */
    unsigned char ucVFD1Pattern;    /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DEC, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern*/
    unsigned short uiVFD2Offset;    /* in bytes */
    unsigned char ucVFD2Range;      /* in bytes */
    unsigned char ucVFD2Pattern;    /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DEC, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern*/
    unsigned short uiVFD3Offset;    /* in bytes */
    unsigned short uiVFD3Range;      /* in bytes */
    unsigned char ucVFD3Enable;      /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField;       /* 0 = off, 1 = insert Signature */
    /* field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6];     /* the Stream's Source MAC addr */
    unsigned char TypeOfService;    /* */
    unsigned char TimeToLive;       /* number of "hops" until frame */
    /* will be dropped */
    unsigned char InitialSequenceNumber; /* Initial sequence number*/
    unsigned char DestinationIP[4]; /* Dest IP addr(e.g. 192.100.5.3) */
    unsigned char SourceIP[4];      /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4];       /* Network Mask (e.g. 255.255.0.0)*/
    unsigned char Gateway[4];       /* Gateway addr (e.g. 192.100.1.1)*/
    unsigned char Protocol;         /* 4=IP on the IP assigned list */
    unsigned char extra[17];        /* reserved */
    unsigned short uiActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPStart;       /* Return value for the Time of */
    /* the last ARP initiated */
    unsigned long ulARPEnd;         /* Return value for the Time of */
    /* the last ARP completed */
    unsigned long ulARPGap;         /* The Time between ARPs */
} StreamIP;

Comment
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>L3_DEFINE_IPX_STREAM</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Create a list with one or more IPX compliant Tx Streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(L3_DEFINE_IPX_STREAM, 0, 0, 0, (void*)pStreamIPX, sizeof(StreamIPX), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**  
StreamIPX

This structure is used to define IPX compliant Streams.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

**NOTE:** The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamIPX
{
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_IPX */
    unsigned char ucReserved; /* Reserved */

    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */

    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern*/

    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern*/

    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */

} StreamIPX;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th>L3_DEFINE_MULTI_IP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Append similar copies of an IP Stream</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(L3_DEFINE_MULTI_IP_STREAM, &lt;index&gt;, &lt;count&gt;, 0, (void*)pStreamIP, sizeof(StreamIP), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>StreamIP</td>
</tr>
</tbody>
</table>

This structure is used to define IP compliant Streams.

The IP checksum is automatically calculated using the supplied header fields and inserted into the IP header.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamIP
{
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_IP */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC */
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DEC, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern */
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DEC, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern */
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField; /* 0 = off, 1 = insert Signature */
    /* field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6]; /* the Stream's Source MAC addr */
    unsigned char TypeOfService; /* */
    unsigned char TimeToLive; /* number of "hops" until frame */
    /* will be dropped */
    unsigned short InitialSequenceNumber; /* Initial sequence number */
    unsigned char DestinationIP[4]; /* Dest IP addr (e.g. 192.100.5.3) */
    unsigned char SourceIP[4]; /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4]; /* Network Mask (e.g. 255.255.0.0) */
    unsigned char Gateway[4]; /* Gateway addr (e.g. 192.100.1.1) */
    unsigned char Protocol; /* 4=IP on the IP assigned list */
    unsigned char extra[17]; /* reserved */
    unsigned long ulActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPStart; /* Return value for the Time of */
    /* the last ARP initiated */
    unsigned long ulARPEnd; /* Return value for the Time of */
    /* the last ARP completed */
    unsigned long ulARPGap; /* The Time between ARPs */
} StreamIP;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_DEFINE_MULTI_IPX_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Append similar copies of an IPX Stream</td>
</tr>
</tbody>
</table>
| **Usage** | ```
int HTSetStructure(L3_DEFINE_MULTI_IPX_STREAM,
                   <index>, <count>, 0,
                   (void*)pStreamIPX,
                   sizeof(StreamIPX),
                   iHub, iSlot, iPort);
``` |
| **Related Structure** | StreamIPX |

This structure is used to define IPX compliant Streams.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

**NOTE:** The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamIPX
{
    unsigned char ucActive;  /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType;  /* use STREAM_PROTOCOL_IPX */
    unsigned char ucRandomLength;  /* Reserved */
    unsigned char ucRandomData;  /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength;  /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset;  /* in bits */
    unsigned char ucVFD1Range;  /* in bits */
    unsigned char ucVFD1Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount;  /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6];  /* the initial VFD byte pattern*/
    unsigned short uiVFD2Offset;  /* in bits */
    unsigned char ucVFD2Range;  /* in bits */
    unsigned char ucVFD2Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount;  /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6];  /* the initial VFD byte pattern*/
    unsigned short uiVFD3Offset;  /* in bytes */
    unsigned short uiVFD3Range;  /* in bytes */
    unsigned char ucVFD3Enable;  /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField;  /* 0 = off, 1 = insert signature */
    /* field into each frame */
    unsigned char DestinationMAC[6];  /* the Stream's Dest MAC addr*/
    unsigned char SourceMAC[6];  /* the Stream's Source MAC addr */
    unsigned short IPXlen;  /* Length field in the IPX hdr */
    unsigned char IPXhop;  /* Hop field in the IPX hdr */
    unsigned char IPXtype;  /* IPX type field in the IPX hdr*/
    unsigned char IPXdst[4];  /* Dest ID in the IPX hdr */
    unsigned char IPXdstHost[6];  /* Dest host in the IPX hdr */
    unsigned short IPXdstSocket;  /* Dest Socket in the IPX hdr */
    unsigned char IPXsrc[4];  /* Source ID in the IPX hdr */
    unsigned char IPXsrcHost[6];  /* Source host in the IPX hdr */
    unsigned short IPXsrcSocket;  /* Source socket in the IPX hdr */
    unsigned char extra[24];  /* reserved */
} StreamIPX;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>lType1</th>
<th>L3_DEFINE_MULTI_SMARTBITS_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Append similar copies of a SmartBits Stream</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(L3_DEFINE_MULTI_SMARTBITS_STREAM, &lt;index&gt;, &lt;count&gt;, 0, (void*)pStreamSmartBits, sizeof(StreamSmartBits), iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

**Related Structure**  
StreamSmartBits

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern. Then define the ProtocolHeader array. This array (of up to 64 bytes) is used similarly to VFD3. It overwrites the background pattern at the specified offset and range. (ucVFD3Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits
{
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    /*************************************************************************/
    /* For ETHERNET, cards VFD1 and VFD2 structure members are */
    /* reserved for later use. Set to 0. */
    /**************************************************************************/
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /*the initial VFD byte pattern*/
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /*the initial VFD byte pattern*/
    /**************************************************************************/
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes; Number of elements */
    /* to use from ProtocolHeader */
    /* No elements are used beyond */
    /* the single specified range. */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField; /* 0 = off, 1 = insert signature*/
    /* field into each frame */
    /**************************************************************************/
    unsigned char ProtocolHeader[64]; /* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_DEFINE_MULTI_UDP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Append similar copies of a UDP Stream</td>
</tr>
</tbody>
</table>
| **Usage**           | ```c
int HTSetStructure(L3_DEFINE_MULTI_UDP_STREAM,
INDEX, COUNT, 0,
(void*)pStreamUDP,
sizeof(StreamUDP),
iHub, iSlot, iPort);
``` |
| **Related Structure** | StreamUDP |

This structure is used to define UDP compliant Streams.

The IP checksums are automatically calculated using the supplied field value and inserted into the frame.

The UDP checksum is set to 0, and so is not calculated.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

**NOTE:** The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamUDP {
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_UDP */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern*/
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern*/
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField; /* 0 = off, 1 = insert signature */
    /* field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6]; /* the Stream's Source MAC addr */
    unsigned char TypeOfService; /* */
    unsigned char TimeToLive; /* number of "hops" until frame */
    /* will be dropped */
    unsigned char InitialSequenceNumber; /* Initial sequence number*/
    unsigned char DestinationIP[4]; /* Dest IP addr(e.g. 192.100.5.3) */
    unsigned char SourceIP[4]; /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4]; /* Network Mask (e.g. 255.255.0.0) */
    unsigned char Gateway[4]; /* Gateway addr (e.g. 192.100.1.1) */
    unsigned short UDPSrc; /* UDP Source Port */
    unsigned short UDPDest; /* UDP Dest Port */
    unsigned short UDPLen; /* UDP Length field */
    unsigned char extra[12]; /* reserved */
    unsigned short uiActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPStart; /* Return value for the Time of */
    /* the last ARP initiated */
    unsigned long ulARPEnd; /* Return value for the Time of */
    /* the last ARP completed */
    unsigned long ulARPGap; /* The Time between ARPs */
} StreamUDP;
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>L3_DEFINE_SMARTBITS_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Create a list with one or more customizable Streams.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetStructure(L3_DEFINE_SMARTBITS_STREAM, 0, 0, 0, (void*)pStreamSmartBits, sizeof(StreamSmartBits), iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

**Related Structure** StreamSmartBits

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern.
Then define the ProtocolHeader array. This array (of up-to 64 bytes) is used similarly to VFD3. It overwrites the background pattern at the specified offset and range.
(ucVFD3Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits
{
    unsigned char ucActive;    /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC */
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */

    /*****************************************************************
    * For ETHERNET, cards VFD1 and VFD2 structure members are
    * reserved for later use. Set to 0.
    */
    /*****************************************************************/
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range;    /* in bits */
    unsigned char ucVFD1Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount;/* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6];/*the initial VFD byte pattern*/

    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range;    /* in bits */
    unsigned char ucVFD2Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount;/* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6];/*the initial VFD byte pattern*/

    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range;  /* in bytes; Number of elements */
    /* to use from ProtocolHeader */
    /* No elements are used beyond */
    /* the single specified range. */
    unsigned char ucVFD3Enable;  /* HVFD_ENABLED, HVFD_NONE */

    unsigned char ucTagField;    /* 0 = off, 1 = insert signature*/
    /* field into each frame */

    unsigned char ProtocolHeader[64]; /* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_DEFINE_UDP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Create a list with one or more UDP compliant Tx Streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetStructure(L3_DEFINE_UDP_STREAM, 0, 0, 0, (void*)pStreamUDP, sizeof(StreamUDP), iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>StreamUDP</td>
</tr>
</tbody>
</table>

This structure is used to define UDP compliant Streams.

The IP checksums are automatically calculated using the supplied field value and inserted into the frame.

The UDP checksum is set to 0, and so is not calculated.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

**NOTE:** The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
```c
typedef struct tagStreamUDP
{
    unsigned char ucActive;  /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_UDP */
    unsigned char ucRandomLength; /* Reserved */

    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
                               /* cards background pattern */
    unsigned char ucRandomData; /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
                               /* 0 - 2048 for Ethernet */
                               /* 0 - 8196 for Frame Relay */

    unsigned short uiVFD1Offset; /* in bits                      */
    unsigned char ucVFD1Range;    /* in bits                      */
    unsigned char ucVFD1Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
                               /* HVFD_INCR, HVFD_DECR, */
                               /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount;/* from 0(off) to 16,777,215 */
                                     /* number to incr. or decr. */
                                     /* through when using inc or */
                                     /* dec pattern */
    unsigned char ucVFD1StartVal[6];/*the initial VFD byte pattern*/

    unsigned short uiVFD2Offset; /* in bits                      */
    unsigned char ucVFD2Range;    /* in bits                      */
    unsigned char ucVFD2Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
                               /* HVFD_INCR, HVFD_DECR, */
                               /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount;/* from 0(off) to 16,777,215 */
                                     /* number to incr. or decr. */
                                     /* through when using inc or */
                                     /* dec pattern */

    unsigned char ucVFD1Enable;   /* HVFD_ENABLED, HVFD_NONE */

    unsigned char ucTagField;     /* 0 = off, 1 = insert signature */
    unsigned char ucTagField;     /* field into each frame */

    unsigned char DestinationMAC[6];/* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6];    /* the Stream's Source MAC addr */
    unsigned char TypeOfService;   /* */
    unsigned char TimeToLive;      /* number of "hops" until frame */
    unsigned char InitialSequenceNumber; /* Initial sequence number*/
    unsigned char DestinationIP[4];/* Dest IP addr(e.g. 192.100.5.3) */
    unsigned char SourceIP[4];     /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4];      /* Network Mask (e.g. 255.255.0.0)*/
    unsigned short Gateway[4];     /* Gateway addr (e.g. 192.100.1.1)*/
    unsigned short UDPSrc;         /* UDP Source Port */
    unsigned short UDPPort;        /* UDP Dest Port */
    unsigned short UDPLen;         /* UDP Length field */
    unsigned char extra[12];       /* reserved */
    unsigned char extra[12];       /* reserved */
    unsigned char uActualSequenceNumber;/* Actual Sequence number */
    unsigned long ulARPStart;      /* Return value for the Time of */
                                      /* the last ARP initiated */
    unsigned long ulARPEnd;        /* Return value for the Time of */
                                      /* the last ARP completed */
    unsigned long ulARPGap;        /* The Time between ARPs */
} StreamUDP;
```
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.

<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_MOD_IP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Overwrite one IP Stream in the list at a given index</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(L3_MOD_IP_STREAM, &lt;index&gt;, 0, 0, (void*)pStreamIP, \ sizeof(StreamIP), \iHub, \iSlot, \iPort) ;</td>
</tr>
</tbody>
</table>

**Related Structure** StreamIP

This structure is used to define IP compliant Streams.

The IP checksum is automatically calculated using the supplied header fields and inserted into the IP header.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamIP {
  /* tags */
  unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
  unsigned char ucProtocolType; /* use STREAM_PROTOCOL_IP */
  unsigned char ucRandomLength; /* Reserved */
  unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
  /* cards background pattern */
  /* not available on Frame Relay */
  unsigned short uiFrameLength; /* frame length not counting CRC */
  /* 0 - 2048 for Ethernet */
  /* 0 - 8196 for Frame Relay */
  unsigned short uiVFD1Offset; /* in bits */
  unsigned char ucVFD1Range; /* in bits */
  unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
  /* HVFD_INCR, HVFD_DECR, */
  /* HVFD_RANDOM, HVFD_NONE */
  unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
  /* number to incr. or decr. */
  /* through when using inc or */
  /* dec pattern */
  unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern */
  unsigned short uiVFD2Offset; /* in bits */
  unsigned char ucVFD2Range; /* in bits */
  unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
  /* HVFD_INCR, HVFD_DECR, */
  /* HVFD_RANDOM, HVFD_NONE */
  unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
  /* number to incr. or decr. */
  /* through when using inc or */
  /* dec pattern */
  unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern */
  unsigned short uiVFD3Offset; /* in bytes */
  unsigned short uiVFD3Range; /* in bytes */
  unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
  unsigned char ucTagField; /* 0 = off, 1 = insert Signature */
  /* field into each frame */
  unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
  unsigned char SourceMAC[6]; /* the Stream's Source MAC addr */
  unsigned char TimeOfService; /* */
  unsigned char TimeToLive; /* number of "hops" until frame */
  /* will be dropped */
  unsigned short InitialSequenceNumber; /* Initial sequence number */
  unsigned char DestinationIP[4]; /* Dest IP addr (e.g. 192.100.5.3) */
  unsigned char SourceIP[4]; /* Src IP addr (e.g. 192.100.5.4) */
  unsigned char Netmask[4]; /* Network Mask (e.g. 255.255.0.0) */
  unsigned char Gateway[4]; /* Gateway addr (e.g. 192.100.1.1) */
  unsigned char Protocol; /* 4=IP on the IP assigned list */
  unsigned char extra[17]; /* reserved */
  unsigned long ulARPStart; /* Actual Sequence number */
  unsigned long ulARPStart; /* Return value for the Time of */
  /* the last ARP completed */
  unsigned long ulARPEnd; /* Return value for the Time of */
  unsigned long ulARPGap; /* The Time between ARPs */
} StreamIP;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>L3_MOD_IPX_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Overwrite one IPX Stream in the list at a given index</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(L3_MOD_IPX_STREAM,</td>
</tr>
<tr>
<td></td>
<td>&lt;index&gt;, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pStreamIPX,</td>
</tr>
<tr>
<td></td>
<td>sizeof(StreamIPX),</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Related Structure</td>
<td>StreamIPX</td>
</tr>
</tbody>
</table>

This structure is used to define IPX compliant Streams.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

NOTE: The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamIPX
{
    unsigned char ucActive; /* 1 = Enable Stream, 0 = Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_IPX */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC */
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    /*****************************************************************
    * For ETHERNET cards, VFD1, VFD2, and VFD3 structure members *
    * are reserved for later use. Set to 0.                         *
    *****************************************************************/
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern */
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern */
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    /******************************************************************/
    unsigned char ucTagField; /* 0 = off, 1 = insert signature */
    /* field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6]; /* the Stream's Source MAC addr */
    unsigned short IPXlen; /* Length field in the IPX hdr */
    unsigned char IPXhop; /* Hop field in the IPX hdr */
    unsigned char IPXtype; /* IPX type field in the IPX hdr */
    unsigned char IPXdst[4]; /* Dest ID in the IPX hdr */
    unsigned char IPXdstHost[6]; /* Dest host in the IPX hdr */
    unsigned short IPXdstSocket; /* Dest Socket in the IPX hdr */
    unsigned char IPXsrc[4]; /* Source ID in the IPX hdr */
    unsigned char IPXsrcHost[6]; /* Source host in the IPX hdr */
    unsigned short IPXsrcSocket; /* Source socket in the IPX hdr */
    unsigned char extra[24]; /* reserved */
} StreamIPX;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th>L3_MOD_SMARTBITS_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Overwrite one SmartBits Stream in the list at a given index</td>
</tr>
</tbody>
</table>
| **Usage** | int HTSetStructure(L3_MOD_SMARTBITS_STREAM, 
<index>, 0, 0, 
(void*)pStreamSmartBits, 
sizeof(StreamSmartBits), 
iHub, iSlot, iPort) ; |
| **Related Structure** | StreamSmartBits |

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern. Then define the ProtocolHeader array. This array (of up to 64 bytes) is used similarly to VFD3. It overwrites the background pattern at the specified offset and range. (ucVFD3Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits
{
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /*the initial VFD byte pattern*/
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /*the initial VFD byte pattern*/
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes; Number of elements */
    /* to use from ProtocolHeader */
    /* No elements are used beyond */
    /* the single specified range. */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField; /* 0 = off, 1 = insert signature*/
    /* field into each frame */
    unsigned char ProtocolHeader[64]; /* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
**iType1**  |  L3_MOD_STREAMS_ARRAY  
---|---  
**Description** | Modify a field in a block of streams by overlaying an element from an array  
**Usage** | int HTSetStructure(L3_MOD_STREAMS_ARRAY,  
0, 0, 0,  
(void*)pLayer3ModifyStreamArray,  
sizeof(Layer3ModifyStreamArray),  
iHub, iSlot, iPort) ;  
**Related Structure** | Layer3ModifyStreamArray  

For use with IP and UDP streams only.

This structure is used to OVERWRITE up to four bytes of a specified field in a block of Streams already defined on the SmartCard.

* Identify the first stream to be modified (ulIndex).
* The total number of streams to be modified is calculated by the number of array elements to use, multiplied by the number of times in a row each element will be repeated: (ulCount * ulFieldRepeat).
* The byte patterns are defined in the elements of ulData[] array. Each pattern is four bytes long. One full pattern per stream is written into the specified field from the least significant bit.
* The field to be modified is specified by ulField. The accepted field values are listed below. If the field is smaller than four bytes, fewer than four bytes will be overwritten.

**NOTE:** For MAC addresses, the least significant four bytes are overwritten. Since the count starts with 0, that would mean bytes 2, 3, 4, and 5 are overwritten.

L3MS_FIELD_DMAC  0 Dest MAC address  
L3MS_FIELD_SMAC  1 Source MAC address  
L3MS_FIELD_TTL  2 Time to Live field  
L3MS_FIELD_ISEQNUM  3 Initial Sequence Number field  
L3MS_FIELD_DIP  4 Dest IP address  
L3MS_FIELD_SIP  5 Source IP address  
L3MS_FIELD_NETMASK  6 The Netmask field  
L3MS_FIELD_GATEWAY  7 First Gateway addr in the Stream struct  
L3MS_FIELD_SPRT  8 Source port number  
L3MS_FIELD_DPRT  9 Dest port number  
L3MS_FIELD_FRAMELEN  10  
L3MS_FIELD_DIPA  11 First dest IP addr in the Stream struct  
L3MS_FIELD_DIPB  12 Second dest IP addr in the Stream struct  
L3MS_FIELD_DIPC  13 Third dest IP addr in the Stream struct  
L3MS_FIELD_DIPD  14  
L3MS_FIELD_DIPA  15 First source IP addr in the Stream struct  
L3MS_FIELD_DIPB  16 Second source IP addr in the Stream struct  
L3MS_FIELD_DIPC  17 Third source IP addr in the Stream struct  
L3MS_FIELD_DIPD  18  
L3MS_FIELD_GATEWAYA  19 Second Gateway addr in the Stream struct  
L3MS_FIELD_GATEWAYB  20 Third Gateway addr in the Stream struct  
L3MS_FIELD_GATEWAYC  21 Fourth Gateway addr in the Stream struct  
L3MS_FIELD_GATEWAYD  22
Example 1:

ulIndex = 2 (start here)
ulCount = 3 (total number of patterns to use)
ulField = L3MS_FIELD_TTL (Time to Live field)
ulFieldCount = 4 (no. of patterns (elements) to use from array)
ulFieldRepeat = 2 (number of times to repeat pattern)
uData [5] [7] [99] [100] (pattern definitions)

(Total fields changed ulCount * ulFieldRepeat = 6)

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>TTL-Value</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Example 2:

ulIndex = 2 (start here)
ulCount = 4 (total number of patterns to use)
ulField = L3MS_FIELD_TTL (Time to Live field)
ulFieldCount = 2 (no. of patterns (elements) to use from array)
ulFieldRepeat = 2 (number of times to repeat pattern)
uData [5] [7] [99] [100] (pattern definitions)

(Total fields changed ulCount * ulFieldRepeat = 8)

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>TTL-Value</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

typedef struct tagLayer3ModifyStreamArray
{ unsigned long ulIndex; /* Index of the first stream to modify */
  unsigned long ulCount; /* The total number of patterns to use */
  /* If ulCount is larger than ulFieldCount, */
  /* patterns will be repeated. */
  unsigned long ulField; /* Field in the stream structure to mod */
  /* See L3MS_FIELD_n defines */
  unsigned long ulFieldCount;
  /* Number of patterns/elements to use */
  /* from uData[] array */
  unsigned long ulFieldRepeat;
  /* number of times each pattern will be */
  /* repeated in a row. */
  /* ulFieldRepeat = 1 does the same thing */
  /* as ulFieldRepeat = 0 */
  unsigned long uData[L3_MODIFY_STREAM_ARRAY_SIZE];
  /* Array of 1,2, or 4 byte patterns/elements */
  /* that will be overlaid (right justified) */
  } Layer3ModifyStreamArray;
**IType1** | L3_MOD_STREAMS_DELTA  
---|---  
**Description** | Increment a field in a block of streams by incrementing a base value  
**Usage** | int HTSetStructure(L3_MOD_STREAMS_DELTA,  
0, 0, 0,  
(void*)pLayer3ModifyStreamDelta,  
sizeof(Layer3ModifyStreamDelta),  
iHub, iSlot, iPort) ;  
**Related Structure** | Layer3ModifyStreamDelta  
For use with IP and UDP streams only.  
This structure is used to INCREMENT one particular field in a block of Streams already defined on the SmartCard. The first field is used as base and is not incremented. The following fields are incremented based on the first field.  
* Identify the stream to use as base. (ulIndex).  
* The total number of streams to be modified is calculated by the number incremented values, multiplied by the number of times in a row each value is repeated: (uiCount * ulFieldRepeat). The base is included in this number as a delta of 0.  
* The amount to increment each field is defined by ulDelta.  
* The field to be modified is specified by ulField. The legal field values are listed below.

| L3MS_FIELD_DMAC | 0 | Dest MAC address  
| L3MS_FIELD_SMAC | 1 | Source MAC address  
| L3MS_FIELD_TTL | 2 | Time to Live field  
| L3MS_FIELD_ISEQNUM | 3 | Initial Sequence Number field  
| L3MS_FIELD_DIP | 4 | Dest IP address  
| L3MS_FIELD_SIP | 5 | Source IP address  
| L3MS_FIELD_NETMASK | 6 | The Netmask field  
| L3MS_FIELD_GATEWAY | 7 | First Gateway addr in the Stream struct  
| L3MS_FIELD_SPRT | 8 | Source port number  
| L3MS_FIELD_DPRT | 9 | Dest port number  
| L3MS_FIELDFRAMELEN | 10 |  
| L3MS_FIELD_DIPA | 11 | First dest IP addr in the Stream struct  
| L3MS_FIELD_DIPB | 12 | Second dest IP addr in the Stream struct  
| L3MS_FIELD_DIPC | 13 | Third dest IP addr in the Stream struct  
| L3MS_FIELD_DIPD | 14 |  
| L3MS_FIELD_SIPA | 15 | First source IP addr in the Stream struct  
| L3MS_FIELD_SIPB | 16 | Second source IP addr in the Stream struct  
| L3MS_FIELD_SIPC | 17 | Third source IP addr in the Stream struct  
| L3MS_FIELD_SIPD | 18 |  
| L3MS_FIELD_GATEWAYA | 19 | Second Gateway addr in the Stream struct  
| L3MS_FIELD_GATEWAYB | 20 | Third Gateway addr in the Stream struct  
| L3MS_FIELD_GATEWAYC | 21 | Fourth Gateway addr in the Stream struct  
| L3MS_FIELD_GATEWAYD | 22 |  

Example 1:

ulIndex = 2 (base stream)
ulCount = 3 (total number of incremented values to use)
ulField = L3MS_FIELD_TTL/TimeToLive (field in structure to increment)
ulFieldRepeat = 2 (number of times to repeat an incremented value)
ulDelta = 3 (amount to increment the field)

(Total fields changed ulCount * ulFieldRepeat = 6)

Before                  After
Index TTL-Value     Index    TTL-Value
1      1             1       1
2      1             2       1 (base value)
3      1             3       1 (repeat base twice)
4      1             4       4 (increment base by 3)
5      1             5       4 (repeat)
6      1             6       7 (increment previous no by 3)
7      1             7       7 (repeat)
8      1             8       1
9      1             9       1
10     1            10       1

Example 2:

ulIndex = 2 (base stream)
ulCount = 3 (total number of incremented values to use)
ulField = L3MS_FIELD_TTL/TimeToLive (field in structure to increment)
ulFieldRepeat = 2 (number of times to repeat an incremented value)
ulDelta = 3 (amount to increment the field)

(Total fields changed ulCount * ulFieldRepeat = 6)

(Difference between example 1 and 2 - initial field Values)

Before                  After
Index TTL-Value     Index    TTL-Value
1      1             1       1
2      2             2       2 (base value)
3      2             3       2 (repeat base twice)
4      2             4       5 (increment base by 3)
5      10            5       5 (repeat)
6      10            6       8 (increment previous no by 3)
7      10            7       8 (repeat)
8      10            8       10
9      10            9       10
10     10            10      10

typedef struct tagLayer3ModifyStreamDelta
{
    unsigned long ulIndex; /* The stream to use as a base. */
    /* This stream is not incremented. */
    /* The next one is incremented, unless */
    /* ulFieldRepeat is greater than 1 */
    unsigned long ulCount; /* The number of different increment */
    /* values to use. */
    unsigned long ulField; /* Field in the stream structure to mod */
    /* See L3MS_FIELD_n defines */
    unsigned long ulFieldRepeat;
    /* Number of times each value will be */
    /* repeated in a row. */
    /* ulFieldRepeat = 1 does the same thing */
    /* as ulFieldRepeat = 0 */
    unsigned long ulDelta; /* The amount to increment the field */
} Layer3ModifyStreamDelta;

Comment

This command/Type1 is supported by the ML-7710 SmartCard, firmware version 1.06 and later only.
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.

<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_MOD_UDP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Overwrite one UDP Stream in the list at a given index</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(L3_MOD_UDP_STREAM, &lt;index&gt;, 0, 0, (void*)pStreamUDP, sizeof(StreamUDP), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>StreamUDP</td>
</tr>
</tbody>
</table>

This structure is used to define UDP compliant Streams.

The IP checksums are automatically calculated using the supplied field value and inserted into the frame.

The UDP checksum is set to 0, and so is not calculated.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

**NOTE:** The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamUDP
{
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_UDP */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECr, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /*the initial VFD byte pattern*/
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECr, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /*the initial VFD byte pattern*/
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    /******************************************************************/
    unsigned char ucTagField; /* 0 = off, 1 = insert signature */
    /* field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6]; /* the Stream's Source MAC addr */
    unsigned char TypeOfService; /* */
    unsigned char TimeToLive; /* number of "hops" until frame */
    /* will be dropped */
    unsigned short InitialSequenceNumber; /* Initial sequence number*/
    unsigned char DestinationIP[4]; /* Dest IP addr (e.g. 192.100.5.3) */
    unsigned char SourceIP[4]; /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4]; /* Network Mask (e.g. 255.255.0.0)*/
    unsigned char Gateway[4]; /* Gateway addr (e.g. 192.100.1.1)*/
    unsigned short UDPSrc; /* UDP Source Port */
    unsigned short UDPDest; /* UDP Dest Port */
    unsigned short UDPLen; /* UDP Length field */
    unsigned char extra[12]; /* reserved */
    unsigned short uiActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPStart; /* Return value for the Time of */
    /* the last ARP initiated */
    unsigned long ulARPEnd; /* Return value for the Time of */
    /* the last ARP completed */
    unsigned long ulARPGap; /* The Time between ARPs */
} StreamUDP;
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.

### L3 - HTGetStructure

<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_ARP_TIMES_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Retrieve ARP Times for each Stream in the list</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(L3_ARP_TIMES_INFO,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pULong,</td>
</tr>
<tr>
<td></td>
<td>sizeof(ULong),</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

**No Related Structure**

### iType1 L3_CAPTURE_COUNT_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th>Get the count of captured frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>int HTGetStructure(L3_CAPTURE_COUNT_INFO,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pLayer3CaptureCountInfo,</td>
</tr>
<tr>
<td></td>
<td>sizeof(Layer3CaptureCountInfo),</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

**Related Structure** Layer3CaptureCountInfo

A simple structure which returns the number of frames which are in the capture buffer.

```c
typedef struct tagLayer3CaptureCountInfo
{
    unsigned long ulCount; /* count of frames in Capture buffer */
    Layer3CaptureCountInfo;
} Layer3CaptureCountInfo;
```
### L3_CAPTURE_PACKET_DATA_INFO

**Description**
Get the captured frame

**Usage**
```
int HTGetStructure(L3_CAPTURE_PACKET_DATA_INFO, 
    <index>, 0, 0, 
    (void*)pLayer3CaptureData, 
    sizeof(Layer3CaptureData), 
    iHub, iSlot, iPort); 
```

**Related Structure**
Layer3CaptureData

Layer3CaptureData is used to retrieve a captured frame from the capture buffer. The length of the frame is given in uiLength, and the captured frame data is put in cData.

```c
typedef struct tagLayer3CaptureData 
{ 
    unsigned short uiLength;   /* the number of bytes in cData which represent the captured frame */ 
    char cData[2048];            /* the captured frame data */ 
} Layer3CaptureData; 
```

### L3_DEFINED_STREAM_COUNT_INFO

**Description**
retrieve the count of Streams in the list

**Usage**
```
int HTGetStructure(L3_DEFINED_STREAM_COUNT_INFO, 
    0, 0, 0, 
    (void*)pULong, 
    sizeof(ULong), 
    iHub, iSlot, iPort); 
```

No Related Structure

**Comment**
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>L3_HIST_ACTIVE_TEST_INFO</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get the number of histogram records, and active histogram</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(L3_HIST_ACTIVE_TEST_INFO, 0, 0, 0, (void*)pLayer3HistActiveTest, sizeof(Layer3HistActiveTest), iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>Layer3HistActiveTest</td>
</tr>
</tbody>
</table>

Used to retrieve the total number of records generated by the active histogram on the specified port.

It also reports which histogram is active.

Histogram records cease to be generated when:
- Information is retrieved from the port using `L3_HIST_ACTIVE_TEST_INFO`.
- Frames with signatures are no longer received.
- Histogram records are retrieved from the port.

```c
typedef struct tagLayer3HistActiveTest
{
    unsigned long ulTest;
    unsigned long ulRecords;
} Layer3HistActiveTest;
```

**Comment**

For related commands and detailed instructions for Histogram results, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_HIST_LATENCY_DISTRIBUTION_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Latency Distribution histogram results</td>
</tr>
</tbody>
</table>

| **Usage** | int HTGetStructure(L3_HIST_LATENCY_DISTRIBUTION_INFO, <index>, 0, 0, (void*)pLayer3StreamDistributionInfo, sizeof(Layer3StreamDistributionInfo), iHub, iSlot, iPort); |

**Related Structure**  
Layer3StreamDistributionInfo

Contains Latency Distribution histogram results.

Latency Distribution tracks how many frames fall within a given latency range. There are 16 configurable ranges in units of .1 microseconds.

**Example Values**

---------------
If `uiInterval[0]` is set to 1, it tracks frames with a latency value of 0 to .1 us. If the next interval is 10, the range is from .2 us to 1 us.

**Mechanics**

--------
When a frame is received, the latency is calculated from the timestamp in the signature field. The correct `ulFrame[i]` is incremented by one. A separate record containing the 16 different range tallies is generated for each stream.

**Resolution and accuracy**

----------
One unit is .1 microsecond - 100ns (one clock tick).

For a pair of 10Mbit ports accuracy is plus or minus .2 microseconds (.2 us).
For a pair of 10/100Mbit ports transmitting at 100Mbits, accuracy is plus or minus .2 us.
For a pair of 10/100Mbit ports transmitting at 10Mbits, accuracy is plus or minus .4 us.
For 10/100Mbit ports transmitting to a 10Mbit port, accuracy is plus or minus .3 us.

**Setting Resolution (different structure)**

----------------------------------------
To set the distribution intervals, use L3_HIST_LATENCY_DISTRIBUTION. This iType1 uses the Layer3HistDistribution structure, where `uiInterval[16]` contains the different latency ranges.

```c
typedef struct tagLayer3StreamDistributionInfo {
    unsigned long ulStream;     /* a unique Stream identifier */
    unsigned long ulFrames[16]; /* Contains the number of frames */
} Layer3StreamDistributionInfo;
```

**Comment**

For related commands and detailed instructions for Histogram results, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_HIST_RAW_TAGS_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Raw Tags histogram records</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(L3_HIST_RAW_TAGS_INFO, &lt;index&gt;, 0, 0, (void*)pLayer3HistTagInfo, sizeof(Layer3HistTagInfo), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>Layer3HistTagInfo</td>
</tr>
</tbody>
</table>

Contains the signature field information for the Raw Tags histogram.

A separate record is generated for each SmartBits test frame (containing a signature) received at this port.

typedef struct tagLayer3HistTagInfo {
    unsigned long ulStream; /* a unique Stream identifier */
    unsigned long ulSequence; /* frame serial number */
    unsigned long ulTransmitTime; /* timestamp of when this frame left its SmartBits port */
    unsigned long ulReceiveTime; /* timestamp of when this frame was received by this SmartBits port */
} Layer3HistTagInfo;

**Comment**

For related commands and detailed instructions for Histogram results, see Chapter 1 of the Message Functions manual.
### lType1 L3_HIST_SEQUENCE_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Sequence Tracking histogram results</th>
</tr>
</thead>
</table>

### Usage

```c
int HTGetStructure(L3_HIST_SEQUENCE_INFO, <index>, 0, 0, (void*)pLayer3SequenceInfo, sizeof(Layer3SequenceInfo), iHub, iSlot, iPort);
```

### Related Structure

Layer3SequenceInfo

Contains Sequence Tracking histogram results.

Sequence Tracking counts the number of frames per stream that are:
* In sequence
* Duplicate
* Lost

A separate record is generated for each test stream received by the specified port.

Sequence is tracked using the signature field embedded in each test frame.

The Sequence Tracking algorithm is as follows:

- As long as frames are received in sequence, the ulSequenced value 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 ulSequence counter.
- 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 (ulLost). 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, ulLost 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, ulDuplicate is incremented.
- The ulSequenced 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, ulLost incremented by four. New hole three frames long.
1,2,3,9,10,11,4,15,5 - ulDuplicate incremented by one. (5 is counted as a duplicate since the previous hole is no longer tracked).
```

typedef struct tagLayer3SequenceInfo
{
    unsigned long ulStream; /* Stream identifier for this record */
    unsigned long ulFrames; /* no. of frames rcvd for this Stream */
    unsigned long ulSequenced; /* no. of frames rcvd in sequence */
    unsigned long ulDuplicate; /* no. of frames duplicated */
    unsigned long ulLost; /* no. of frames that broke the */
    /* sequence and were either never rcvd,*/
    /* or not received before another out- */
    /* of-sequence frame was noted. */
} Layer3SequenceInfo;
### Comment

For related commands and detailed instructions for Histogram results, see Chapter 1 of the Message Functions manual.

<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_HIST_V2_LATENCY_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Latency over Time histogram results</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(L3_HIST_V2_LATENCY_INFO, &lt;index&gt;, 0, 0, (void*)pLayer3LongLatencyInfo, sizeof(Layer3LongLatencyInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**  
Layer3LongLatencyInfo

Contains Latency Over Time histogram results.

Provides the Minimum, Maximum, and Sum of Latency values for frames received during specified intervals. The values are the composite results of all streams averaged together.

**Setting the interval**

On the Rx port, use the Layer3HistLatency structure and specify the ulInterval. This value determines how often latency values are compared.

One Interval unit is 1 millisecond.

The Latency measurements of, Minimum, Maximum, and Average Latency are calculated to the nearest .1 microsecond.

**Example**

If ulInterval is set to 10:

The first Min, Max, and Sum latency value is for the first ten microseconds. The second Min, Max, and Sum latency value is for the second ten microseconds, and so on.

Latency values are calculated from the timestamp in the signature field embedded within each test frame.

The average latency for each time interval can be calculated by dividing the total latency by the number of frames.

```c
typedef struct tagLayer3LongLatencyInfo
{
    unsigned long ulMinimum;   /* min frame latency in .1 microseconds */
    unsigned long ulMaximum;   /* max frame latency in .1 microseconds */
    U64 u64Total;              /* sum of all latencies .1 microseconds */
    unsigned long ulFrames;   /* frames with signature fields (tags), */
    /* received in this interval. */
} Layer3LongLatencyInfo;
```

### Comment

For related commands and detailed instructions for Histogram results, see Chapter 1 of the Message Functions manual.
### iType1

<table>
<thead>
<tr>
<th>L3_HIST_V2_LATENCY_PER_STREAM_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Usage</strong></td>
</tr>
</tbody>
</table>

**Related Structure**  
Layer3StreamLongLatencyInfo  
Contains results from a combination histogram, V2 Latency Per Stream.

This histogram tracks:
1. Latency Per Stream (Min, Max, Average)
2. Sequence Tracking
3. Latency Distribution.

One record is generated for each stream received. Statistics are tracked using the signature field embedded within each test frame.

To specify this histogram, use L3_HIST_V2_LATENCY_PER_STREAM.

Latency Resolution and accuracy
---------------------------------
One unit is .1 microsecond / 100ns (one clock tick).

* For a pair of 10Mbit ports, accuracy is plus or minus .2 us.
* For a pair of 10/100Mbit ports transmitting at 100Mbits, accuracy is plus or minus .2 us.
* For a pair of 10/100Mbit ports transmitting at 10Mbits, accuracy is plus or minus .4 us.
* For 10/100Mbit ports transmitting to a 10Mbit port, accuracy is plus or minus .3 us.

**Latency Per Stream Info.**
********************************
Latency Per Stream finds the Sum, Min, and Max latency value for each stream over the duration of the test.

**Sequence Tracking Info.**
********************************
Sequence Tracking counts the number of frames per stream that are:
* Out of sequence
* Duplicate
* Lost

A separate record is generated for each test stream received by the specified port.
The Sequence Tracking algorithm is as follows:

* As long as frames are received in sequence, the ulSequenced value 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 ulSequence counter.
* 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 (ulLost). 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, ulLost 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, ulDuplicate is incremented.
* The ulSequenced 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, ulLost incremented by four. New hole three frames long.
1,2,3,9,10,11,4,15,5 - ulDuplicate incremented by one. (5 is counted as a duplicate since the previous hole is no longer tracked).

Latency Distribution Info.
************************
Latency Distribution tracks the number of frames with a latency that falls within a given range. (There are 16 configurable ranges).

Example Values
--------------
If ulInterval[0] is set to 1, it tracks frames with a latency value of 0 to .1 us. If the next interval is 10, the range is from .2us to 1us.

Mechanics
--------
When a frame is received, correct ulFrame[i] is incremented by one. A separate record containing the 16 different range tallies is generated for each stream.
typedef struct tagLayer3StreamLongLatencyInfo
{
    unsigned long ulStream; /* a unique Stream Identifier */
    /* Latency Per Stream info. */
    U64 u64Total;     /* sum of latencies for this Stream */
    unsigned long ulMinimum; /* min frame latency for this Stream */
    unsigned long ulMaximum; /* max frame latency for this Stream */
    /* Sequenc Tracking info. */
    unsigned long ulTotalFrames; /* total frames rcvd for this Stream */
    unsigned long ulMinimum; /* min frame latency for this Stream */
    unsigned long ulDuplicate; /* no. of frames duplicated */
    /* MC only >> 1st timestamp rcvd for group stream */
    unsigned long ulLost;       /* no. frames that broke the sequence */
    /* or not received before another out-of-sequence frame was noted. */
    /* currently unused for MC */
    /* Latency Distribution info. */
    unsigned long ulFrames[16]; /* Contains the number of frames */
    /* for 16 different ranges */
} Layer3StreamLongLatencyInfo;

Comment

For related commands and detailed instructions for Histogram results, see Chapter 1 of the Message Functions manual.

This structure contains an embedded structure.

<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_STREAM_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Retrieves information about a Stream at the specified index.</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTGetStructure(L3_STREAM_INFO,</td>
</tr>
<tr>
<td></td>
<td>&lt;index&gt;&gt;, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pStreamSmartBits,</td>
</tr>
<tr>
<td></td>
<td>sizeof(StreamSmartBits),</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

Related Structure StreamSmartBits

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern.
Then define the ProtocolHeader array. This array (of up to 64 bytes) is used similarly to VFD3. It overwrites the background pattern at the specified offset and range. (ucVFD3Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits
{
    unsigned char ucActive;       /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData;   /* 1 = Random Data, 0 = use the */
        /* cards background pattern */
        /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
        /* 0 - 2048 for Ethernet */
        /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset;   /* in bits */
    unsigned char ucVFD1Range;     /* in bits */
    unsigned char ucVFD1Pattern;   /* HVFD_ENABLED, HVFD_STATIC, */
        /* HVFD_INCR, HVFD_DEC, */
        /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount;/* from 0(off) to 16,777,215 */
        /* number to incr. or decr. */
        /* through when using inc or */
                /* dec pattern */
    unsigned char ucVFD1StartVal[6];/*the initial VFD byte pattern*/
    unsigned short uiVFD2Offset;   /* in bits */
    unsigned char ucVFD2Range;     /* in bits */
    unsigned char ucVFD2Pattern;   /* HVFD_ENABLED, HVFD_STATIC, */
        /* HVFD_INCR, HVFD_DEC, */
        /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount;/* from 0(off) to 16,777,215 */
        /* number to incr. or decr. */
        /* through when using inc or */
                /* dec pattern */
    unsigned char ucVFD2StartVal[6];/*the initial VFD byte pattern*/
    unsigned short uiVFD3Offset;   /* in bytes */
    unsigned short uiVFD3Range;    /* in bytes; Number of elements */
        /* to use from ProtocolHeader */
        /* No elements are used beyond */
                /* the single specified range. */
    unsigned char ucVFD3Enable;    /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField;      /* 0 = off, 1 = insert signature*/
        /* field into each frame */
    unsigned char ProtocolHeader[64];/* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment
NOTE: This will also get information about the place-holder stream at index 0.

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
### L3 - HTSetCommand

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>L3_CAPTURE_ALL_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Capture all frames received</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(L3_CAPTURE_ALL_TYPE, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>

```c
typedef struct tagLayer3Address
{
    unsigned char szMACAddress[6];  /* sets MAC addr of this SmartCard */
    unsigned char IP[4];           /* sets IP addr of this SmartCard */
    unsigned char Netmask[4];      /* sets Netmask for this SmartCard */
    unsigned char Gateway[4];      /* sets Gateway addr for this Card */
    unsigned char PingTargetAddress[4];  /* the address where PINGs are sent*/
    int iControl;                  /* use L3_CTRL_ defines below */
    /* enables PING, ARP, SNMP or RIP */
    int iPingTime;                 /* in seconds - 0 disables */
    int iSNMPTime;                 /* in seconds - 0 disables */
    int iRIPTime;                  /* in seconds - 0 disables */
    int iGeneralIPResponse;        /* obsolete */
} Layer3Address;
```

---

### L3_TX_ADDRESS_INFO

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>L3_TX_ADDRESS_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>SmartCard device address information</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTGetStructure(L3_TX_ADDRESS_INFO, 0, 0, 0, (void*)pLayer3Address, sizeof(Layer3Address), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>Layer3Address</td>
</tr>
</tbody>
</table>

This structure is used to set the CARD address, as opposed to the stream address. It also sets other L3 information that allows the card to send PING, SNMP, RIP, and ARP frames.

Once set, these frames are transmitted automatically at the specified interval.

To enable these frames, set iControl to:
- L3_CTRL_ARP_RESPONSES 0x01
- L3_CTRL_PING_RESPONSES 0x02
- L3_CTRL_SNMP_OR_RIP_RESPONSES 0x04

---

**Comment**

**L3 - HTSetCommand**

- **iType1**: L3_CAPTURE_ALL_TYPE
- **Description**: Capture all frames received
- **Usage**: int HTSetCommand(L3_CAPTURE_ALL_TYPE, 0, 0, 0, NULL, iHub, iSlot, iPort);
- **No Related Structure**
- **Comment**

**L3_TX_ADDRESS_INFO**

- **iType1**: L3_TX_ADDRESS_INFO
- **Description**: SmartCard device address information
- **Usage**: int HTGetStructure(L3_TX_ADDRESS_INFO, 0, 0, 0, (void*)pLayer3Address, sizeof(Layer3Address), iHub, iSlot, iPort);
- **Related Structure**: Layer3Address

This structure is used to set the CARD address, as opposed to the stream address. It also sets other L3 information that allows the card to send PING, SNMP, RIP, and ARP frames.

Once set, these frames are transmitted automatically at the specified interval.

To enable these frames, set iControl to:
- L3_CTRL_ARP_RESPONSES 0x01
- L3_CTRL_PING_RESPONSES 0x02
- L3_CTRL_SNMP_OR_RIP_RESPONSES 0x04

**Comment**
## L3_CAPTURE_BAD_TYPE

**Description**
Capture errors frames only

**Usage**

```c
int HTSetCommand(L3_CAPTURE_BAD_TYPE,
                 0, 0, 0,
                 NULL,
                 iHub, iSlot, iPort);
```

**Comment**
No Related Structure

## L3_CAPTURE_OFF_TYPE

**Description**
Turn Capture off

**Usage**

```c
int HTSetCommand(L3_CAPTURE_OFF_TYPE,
                 0, 0, 0,
                 NULL,
                 iHub, iSlot, iPort);
```

**Comment**
No Related Structure

## L3_CAPTURE_TRIGGERS_TYPE

**Description**
Capture Trigger frames only

**Usage**

```c
int HTSetCommand(L3_CAPTURE_TRIGGERS_TYPE,
                 0, 0, 0,
                 NULL,
                 iHub, iSlot, iPort);
```

**Comment**
No Related Structure

## L3_HIST_LATENCY_DISTRIBUTION

**Description**
Get Latency Distribution histogram results

**Usage**

```c
int HTSetCommand(L3_HIST_LATENCY_DISTRIBUTION,
                 0, 0, 0,
                 (void*)pLayer3HistDistribution,
                 iHub, iSlot, iPort);
```

**Related Structure**
Layer3HistDistribution
Defines the time interval ranges used in the 16-bit version of the Latency Distribution histogram. The maximum latency it can track is 6.5 milliseconds.

NOTE: For 32 bit Latency Distribution tracking (with latency tracking up to 429.4 seconds) use the combination histogram set by L3_HIST_V2_LATENCY_PER_STREAM.

Latency Distribution tracks how many frames fall within a given latency range. There are 16 configurable ranges in units of .1 microseconds.

Example Values
--------------
If uiInterval[0] is set to 1, it tracks frames with a latency value of 0 to .1 us. If the next interval is 10, the range is from .2us to 1us.

Mechanics
---------
When a frame is received, the latency is calculated from the timestamp in the signature field. The correct ulFrame[i] is incremented by one. A separate record containing the 16 different range tallies is generated for each stream.

Resolution and accuracy
-----------------------
One unit is .1 microsecond (one clock tick).

For a pair of 10Mbit ports accuracy is plus or minus .2 us.
For a pair of 10/100Mbit ports transmitting at 100Mbits, accuracy is plus or minus .2 us.
For a pair of 10/100Mbit ports transmitting at 10Mbits, accuracy is plus or minus .4 us.
For 10/100Mbit ports transmitting to a 10Mbit port, accuracy is plus or minus .3 us.

Results for this histogram are retrieved using the Layer3StreamDistributionInfo structure.

typedef struct tagLayer3HistDistribution
{
    unsigned short uiInterval[16];    /* 1 = 100 nanoseconds */
} Layer3HistDistribution;

Comment

---

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>L3_HIST_RAW_TAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Raw Tags histogram records</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(L3_HIST_RAW_TAGS, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

No Related Structure

Comment

---

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>L3_HIST_SEQUENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Sequence Tracking histogram results</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(L3_HIST_SEQUENCE, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

No Related Structure

Comment
<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_HIST_START</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Clear Histogram records/Histogram remains in Receive state.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(L3_HIST_START, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td>No Related Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>L3_HIST_V2_LATENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Latency over Time histogram results</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(L3_HIST_V2_LATENCY, 0, 0, 0, (void*)pLayer3HistLatency, iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>Layer3HistLatency</td>
</tr>
</tbody>
</table>
| **Comment** | Used to setup time intervals for the Latency Over Time histogram. This structure is used with L3_HIST_V2_LATENCY. The Latency Over Time histogram provides the Minimum, Maximum, and Average Latency for all streams at specified intervals. ulInterval is the interval that latency values are checked. For example, if ulInterval is 2, latency values are calculated every 200 nanoseconds (.2us). Latency Resolution and accuracy
--------------------
One unit is .1 microsecond / 100ns (one clock tick).
* For a pair of 10Mbit ports, accuracy is plus or minus .2 us.
* For a pair of 10/100Mbit ports transmitting at 100Mbits, accuracy is plus or minus .2 us.
* For a pair of 10/100Mbit ports transmitting at 10Mbits, accuracy is plus or minus .4 us.
* For 10/100Mbit ports transmitting to a 10Mbit port, accuracy is plus or minus .3 us.
Results are retrieved using the Layer3LongLatencyInfo structure. |
| \[
\text{typedef struct tagLayer3HistLatency}
\{
    \text{unsigned long ulInterval; \quad /* 1 = 1 millisecond */}
\} Layer3HistLatency;
\] |
### L3_HIST_V2_LATENCY_PER_STREAM

**Description**
Get combination histogram results - Latency Dist, Lat. Per Stream, Sequence.

**Usage**
```c
int HTSetCommand(L3_HIST_V2_LATENCY_PER_STREAM,
                  0, 0, 0,
                  (void*)pLayer3V2HistDistribution,
                  iHub, iSlot, iPort);
```

**Related Structure**
Layer3V2HistDistribution

Defines the time interval ranges used in the combination histogram specified by L3_HIST_V2_LATENCY_PER_STREAM. The maximum latency it can track is 429.4 seconds.

Latency Distribution tracks how many frames have a latency that falls within a given range. There are 16 configurable ranges in units of .1 microseconds.

**Example Values**
-------------
If ulInterval[0] is set to 1, it tracks frames with a latency value of 0 to .1 us. If the next interval is 10, the range is from .2us to 1us.

**Mechanics**
---------
When a frame is received, the latency is calculated from the timestamp in the signature field. The correct ulFrame[i] is incremented by one. A separate record containing the 16 different range tallies is generated for each stream.

**Resolution and accuracy**
-----------------------
One unit is .1 microsecond (one clock tick).

For a pair of 10Mbit ports accuracy is plus or minus .2 us.
For a pair of 10/100Mbit ports transmitting at 100Mbits, accuracy is plus or minus .2 us.
For a pair of 10/100Mbit ports transmitting at 10Mbits, accuracy is plus or minus .4 us.
For a 10/100Mbit port transmitting to a 10Mbit port, accuracy is plus or minus .3 us.

Results for this histogram are retrieved using the Layer3StreamDistributionInfo structure.

```c
typedef struct tagLayer3V2HistDistribution
{
    unsigned long ulInterval[16];        /* 1 = 100 nanoseconds */
} Layer3V2HistDistribution;
```

### L3_START_ARPS

**Description**
Begin ARP exchange on all defined Streams

**Usage**
```c
int HTSetCommand(L3_START_ARPS,
                 0, 0, 0,
                 NULL,
                 iHub, iSlot, iPort);
```

**Comment**

No Related Structure
Chapter 8: Frame Relay

This section covers the Message Functions as Frame Relay SmartCards. This Chapter covers the Frame Relay parameters and structures.

Note: Some structures contain embedded or nested structures. In these cases, the embedded structures are included directly below the related structure.

These commands (iType1s) and related structures work with both WN-3405 and the WN-3415 SmartCards with these exceptions:
FR_LINE works only with WN-3405.
FR_T1E1_LINE works only with WN-3415.

FR - HTSetStructure Summary

<table>
<thead>
<tr>
<th>iTYPE1</th>
<th>iTYPE2</th>
<th>iTYPE3</th>
<th>iTYPE4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_CARD_CFG</td>
<td>0 0 0</td>
<td>0</td>
<td></td>
<td>FRCardCfg</td>
<td>Config global card params</td>
</tr>
<tr>
<td>FR_DEFINE_IP_STREAM</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>StreamIP</td>
<td>Define an IP stream</td>
</tr>
<tr>
<td>FR_DEFINE_SMARTBITS_STREAM</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>StreamSmartBits</td>
<td>raw stream</td>
</tr>
<tr>
<td>FR_DEFINE_UDP_STREAM</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>StreamUDP</td>
<td>Define a UDP stream</td>
</tr>
<tr>
<td>FR_DUP_PVC</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>FRPvcTableEntry</td>
<td>Duplicate an existing PVC config</td>
</tr>
<tr>
<td>FR_DUP_STREAM</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>StreamSmartBits</td>
<td>reserved - duplicate streams</td>
</tr>
<tr>
<td>FR_FILL_PATTER</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>UChar</td>
<td>Config global background fill pattern</td>
</tr>
<tr>
<td>FR_HIST_LATENCY_DISTRIBUTION</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>Layer3HistDistribution</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_SEQUENCE</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_V2_LATENCY</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>Layer3HistLatency</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_V2_LATENCY_PER_STREAM</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>Layer3V2HistDistribution</td>
<td></td>
</tr>
<tr>
<td>FR_IP_SUBNET_DEREG</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRIPSubnetDeRegister</td>
<td>Remove a new or existing IP subnet addr</td>
</tr>
<tr>
<td>FR_IP_SUBNET_REG</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRIPSubnetRegister</td>
<td>Config a new or existing IP subnet addr</td>
</tr>
<tr>
<td>FR_LINE</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRLineCfg</td>
<td>Config physical layer</td>
</tr>
<tr>
<td>FR_LMI</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRLmiCfg</td>
<td>Config LMP signaling timers, counter and mode</td>
</tr>
<tr>
<td>FR_MOD_UDP_STREAM</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>StreamUDP</td>
<td>reserved</td>
</tr>
<tr>
<td>FR_PVC</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRPvcTableEntry</td>
<td>Configure a new or existing PVC</td>
</tr>
<tr>
<td>FR_PVC_CTRL</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRPvcControl</td>
<td>Control a single PVC: disable/enable</td>
</tr>
<tr>
<td>FR_PVC_STREAM_MAP_CFG</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRPvcStrmMapCfg</td>
<td>Configure per stream PVC related params</td>
</tr>
<tr>
<td>FR_STRM_CTRL</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRStreamControl</td>
<td>Control a stream: disable/enable, generate err etc</td>
</tr>
<tr>
<td>FR_T1E1_LINE</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRT1E1LineCfg</td>
<td>Config T1E1 physical layer</td>
</tr>
<tr>
<td>FR_TRIGGER</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td>FRTTriggerCfg</td>
<td>Config global tx and receive triggers</td>
</tr>
</tbody>
</table>
### FR - HTGetStructure Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_AGGR_LATENCY_DISTRIBUTION_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3StreamDistributionInfo</td>
<td>Get Full Latency Distribution histogram results</td>
</tr>
<tr>
<td>FR_AGGR_SEQUENCE_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3SequenceInfo</td>
<td>Get Sequence Tracking histogram results</td>
</tr>
<tr>
<td>FR_AGGR_V2_LATENCY_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3LongLatencyInfo</td>
<td>Get 32-bit Latency over Time histogram results</td>
</tr>
<tr>
<td>FR_AGGR_V2_LATENCY_PER_STREAM_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Layer3StreamLongLatencyInfo</td>
<td>Get 32-bit Latency Distribution and Per Stream histogram results</td>
</tr>
<tr>
<td>FR_CARD_VERSION_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FRVersionInfo</td>
<td>Get card firmware versions</td>
</tr>
<tr>
<td>FR_DEFINED_STREAM_COUNT_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ULong</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_ACTIVE_TEST_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>Layer3HistActiveTest</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_LATENCY_DISTRIBUTION_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>Layer3StreamDistributionInfo</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_SEQUENCE_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>Layer3SequenceInfo</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_TYPE_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Get histogram result data</td>
</tr>
<tr>
<td>FR_HIST_V2_LATENCY_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>Layer3StreamLongLatencyInfo</td>
<td></td>
</tr>
<tr>
<td>FR_HIST_V2_LATENCY_PER_STREAM_INFO</td>
<td>&lt;index&gt;</td>
<td>&lt;count&gt;</td>
<td>0</td>
<td>Layer3StreamLongLatencyInfo</td>
<td></td>
</tr>
<tr>
<td>FR_IP_STREAM_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>StreamIP</td>
<td>Get IP stream config info.</td>
</tr>
<tr>
<td>FR_LINK_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FRLinkInfo</td>
<td>Get link statistics counters</td>
</tr>
<tr>
<td>FR_LINK_STATUS_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FRLinkStatusInfo</td>
<td>Get link status</td>
</tr>
<tr>
<td>FR_LMI_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FRLmiInfo</td>
<td>Get LMP statistics counters</td>
</tr>
<tr>
<td>FR_PVC_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>FRPvcMainInfo</td>
<td>Get per-PVC statistics counters</td>
</tr>
<tr>
<td>FR_PVC_STATUS_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>FRPVCStatusInfo</td>
<td>Get PVC status for all 1024 PVCs</td>
</tr>
<tr>
<td>FR_SMARTBITS_STREAM_INFO</td>
<td>&lt;index&gt;</td>
<td>0</td>
<td>0</td>
<td>StreamSmartBits</td>
<td></td>
</tr>
<tr>
<td>FR_T1E1_LINE_INFO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FRT1E1LineInfo</td>
<td>Get T1E1 physical layer info</td>
</tr>
</tbody>
</table>

### FR - HTSetCommand Summary

<table>
<thead>
<tr>
<th>iType1</th>
<th>iType2</th>
<th>iType3</th>
<th>iType4</th>
<th>pData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_CLEAR_COUNTERS_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Clear all statistic counters</td>
</tr>
<tr>
<td>FR_COMMIT_CFG</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Commit PVC, stream and IP subnet config.</td>
</tr>
<tr>
<td>FR_DISABLE_PORT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Disable WAN port</td>
</tr>
<tr>
<td>FR_ENABLE_PORT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Enable WAN port</td>
</tr>
<tr>
<td>FR_GROUP_MEMBER_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Set card to be a member of group</td>
</tr>
<tr>
<td>FR_GROUP_START_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Start transmission if belong to group</td>
</tr>
<tr>
<td>FR_GROUP_STEP_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Send one frame if belong to group</td>
</tr>
<tr>
<td>FR_GROUP_STOP_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Stop transmission if belong to group</td>
</tr>
<tr>
<td>FR_NON_GROUP_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>unset card from group</td>
</tr>
<tr>
<td>FR_PVC_DELETE_ALL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Delete all existing PVCs</td>
</tr>
<tr>
<td>FR_SET_START_CFG</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Start PVC, stream and IP subnet addr. config</td>
</tr>
<tr>
<td>FR_START_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Start transmission</td>
</tr>
<tr>
<td>FR_STEP_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Sent one frame</td>
</tr>
<tr>
<td>FR_STOP_CMD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Stop transmission</td>
</tr>
<tr>
<td>FR_STREAM_DELETE_ALL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Delete all existing streams</td>
</tr>
</tbody>
</table>
**FR - HTSetStructure**

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_CARD_CFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Config global card params</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_CARD_CFG, 0, 0, 0, (void*)pFRCardCfg, sizeof(FRCardCfg), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**  FRCardCfg

This structure sets the transmit mode, histogram type, latency scale and protocol frames for the card. All local IP addresses can be set here, but IP subnet registering structure should be used.

Note that protocol frames are restarted every time this structure is received. Transmit traffic will continue. The new transmit mode will become effective when test traffic is restarted.

Card configuration is stored in non-volatile memory.

```c
typedef struct tagFRCardCfg {
    unsigned long ulMultiBurstCnt; /* Number of bursts to repeat */
    unsigned long ulBurstCnt; /* Number of frames per burst */
    unsigned long ulInterBurstGap; /* Number of Flags between bursts*/
    unsigned long ulTransmitMode; /* Transmit mode */
        /* FR_TX_CONTINUOUS or FR_TX_SINGLE_BURST or */
        /* FR_TX_MULTI_BURST or FR_TX_CONTINUOUS_MULTI_BURST */
    unsigned char ucCardNum; /* card number */
    unsigned char ucGroupMember; /* boolean TRUE: group member */
    unsigned char ucMACAddress[6]; /* Ethernet MAC address for card: for bridging only */
    unsigned char ucIPAddress[4]; /* IP addr of card if IP routing */
    unsigned char ucNetmask[4]; /* IP netmask */
    unsigned char ucDefaultGateway[4]; /* IP default gateway addr */
    unsigned char ucPingTargetAddress[4]; /* Ping target IP addr. */
    unsigned char ucLatencyScaling; /* Time scale for histogram latency test */
        /* TBD */
    unsigned char ucHistogramType; /* Histogram type */
        /* HIST_OFF or HIST_SEQ_TRACK or HIST_LAT_TIME or */
        /* HIST_LAT_STREAM or HIST_LAT_DISTRIBUTION or */
        /* HIST_RAW_TAGS or HIST_LONG_LAT_TIME or */
        /* HIST_LONG_LAT_STREAM */
    unsigned char ucLmiOn; /* OBSOLETE */
    unsigned char ucSnmpFrames; /* 1 = send periodic SNMP frame */
    unsigned char ucProtocolFrames; /* 1 = send periodic RIP frame */
    unsigned char ucPingFrames; /* 1 = send ping frame */
    unsigned char ucRipPeriod; /* RIP frame cycle in second */
    unsigned char ucSnmpPeriod; /* SNMP frame cycle in second */
    unsigned char ucPingPeriod; /* PING frame cycle in second */
    unsigned char ucGeneralIPResponse; /* TRUE: reply to ALL ARP req. */
        /* not destined to card IP or stream source IP addr */
    unsigned char ucEncapType; /* Default RFC-1490 encapsulation type for card */
        /* FR_NO_ENCAP or */
        /* FR_RFC1490_BRIDGED_SNAP or */
        /* FR_RFC1490_RTD_NLPID or */
    unsigned char ucReserved[17];
} FRCardCfg; /* FR_CARD_CFG_PARAM_T; */
```
**Comment**

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_DEFINE_IP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Define an IP stream</td>
</tr>
</tbody>
</table>
| **Usage**       | int HTSetStructure(FR_DEFINE_IP_STREAM,  
|                 | 0, 0, 0,  
|                 | (void*)pStreamIP,  
|                 | sizeof(StreamIP),  
|                 | iHub, iSlot, iPort) ; |
| **Related Structure** | StreamIP |

This structure is used to define IP compliant Streams.

The IP checksum is automatically calculated using the supplied header fields and inserted into the IP header.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamIP
{
    unsigned char ucActive;    /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType;   /* use STREAM_PROTOCOL_IP */
    unsigned char ucRandomLength;   /* Reserved */

    unsigned char ucRandomData;   /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength;   /* frame length not counting CRC */
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */

    /**************************************************************************
    * For ETHERNET, cards VFD1, VFD2, and VFD3 structure members *
    * are reserved for later use. Set to 0.                          *
    *----------------------------------------------------------------------*
    */
    unsigned short uiVFD1Offset;  /* in bits */
    unsigned char ucVFD1Range;     /* in bits */
    unsigned char ucVFD1Pattern;   /* HVFD_ENABLED, HVFD_STATIC, */
    / HVFD_INCR, HVFD_DEC, */
    / HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6];  /* the initial VFD byte pattern */

    unsigned short uiVFD2Offset;  /* in bits */
    unsigned char ucVFD2Range;     /* in bits */
    unsigned char ucVFD2Pattern;   /* HVFD_ENABLED, HVFD_STATIC, */
    / HVFD_INCR, HVFD_DEC, */
    / HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6];  /* the initial VFD byte pattern */

    unsigned short uiVFD3Offset;  /* in bytes */
    unsigned short uiVFD3Range;    /* in bytes */
    unsigned char ucVFD3Enable;    /* HVFD_ENABLED, HVFD_NONE */

    /**************************************************************************
    unsigned char ucTagField;  /* 0 = off, 1 = insert Signature */
    / field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6];      /* the Stream's Source MAC addr */
    unsigned char TypeOfService;  /* */
    unsigned char TimeToLive;   /* number of "hops" until frame */
    / will be dropped */
    unsigned short InitialSequenceNumber; /* Initial sequence number*/
    unsigned char DestinationIP[4]; /* Dest IP addr(e.g. 192.100.5.3) */
    unsigned char SourceIP[4];      /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4];       /* Network Mask (e.g. 255.255.0.0) */
    unsigned char Gateway[4];      /* Gateway addr (e.g. 192.100.1.1) */
    unsigned char Protocol;         /* 4=IP on the IP assigned list */
    unsigned char extra[17];       /* reserved */
    unsigned short uiActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPSend;       /* Return value for the Time of */
    / the last ARP completed */
    unsigned long ulARPStart;      /* Return value for the Time of */
    / the last ARP initiated */
    unsigned long ulARPGap;        /* The Time between ARPs */

}| StreamIP:

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>FR_DEFINE_SMARTBITS_STREAM</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>raw stream</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FR_DEFINE_SMARTBITS_STREAM, 0, 0, 0, (void*)pStreamSmartBits, sizeof(StreamSmartBits), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**  StreamSmartBits

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern. Then define the ProtocolHeader array. This array (of up-to 64 bytes) is used similarly to VFD3. It overwrites the background pattern at the specified offset and range. (ucVFD3Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits
{
    unsigned char ucActive;    /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData;  /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */

    /***************************************************************************/
    /* If ETHERNET, cards VFD1 and VFD2 structure members are */
    /* reserved for later use. Set to 0. */
    /***************************************************************************/
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range;   /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DEC, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6];/*the initial VFD byte pattern*/

    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range;   /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DEC,*/
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6];/*the initial VFD byte pattern*/

    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range;  /* in bytes; Number of elements */
    /* to use from ProtocolHeader */
    /* No elements are used beyond */
    /* the single specified range. */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */

    unsigned char ucTagField;    /* 0 = off, 1 = insert signature*/
    /* field into each frame */

    unsigned char ProtocolHeader[64];/* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_DEFINE_UDP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Define a UDP stream</td>
</tr>
<tr>
<td>Usage</td>
<td><code>int HTSetStructure(FR_DEFINE_UDP_STREAM,</code></td>
</tr>
<tr>
<td></td>
<td><code>0, 0, 0,</code></td>
</tr>
<tr>
<td></td>
<td><code>(void*)pStreamUDP,</code></td>
</tr>
<tr>
<td></td>
<td><code>sizeof(StreamUDP),</code></td>
</tr>
<tr>
<td></td>
<td><code>iHub, iSlot, iPort)</code>;</td>
</tr>
<tr>
<td>Related Structure</td>
<td>StreamUDP</td>
</tr>
</tbody>
</table>

This structure is used to define UDP compliant Streams.

The IP checksums are automatically calculated using the supplied field value and inserted into the frame.

The UDP checksum is set to 0, and so is not calculated.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

NOTE: The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamUDP
{
  unsigned char ucActive;         /* 1=Enable Stream, 0=Disable Stream */
  unsigned char ucProtocolType;   /* use STREAM_PROTOCOL_UDP */
  unsigned char ucRandomLength;   /* Reserved */

  unsigned char ucRandomData;     /* 1 = Random Data, 0 = use the */
  /* cards background pattern */
  /* not available on Frame Relay */
  unsigned short uiFrameLength;   /* frame length not counting CRC*/
  /* 0 - 2048 for Ethernet */
  /* 0 - 8196 for Frame Relay */

  unsigned short uiVFD1Offset;    /* in bits                      */
  unsigned char ucVFD1Range;      /* in bits                      */
  unsigned char ucVFD1Pattern;    /* HVFD_ENABLED, HVFD_STATIC, */
  /* HVFD_INCR, HVFD_DEC, */
  /* HVFD_RANDOM, HVFD_NONE */
  unsigned long ulVFD1PatternCount;/* from 0(off) to 16,777,215 */
  /* number to incr. or decr. */
  /* through when using inc or */
  /* dec pattern */
  unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern*/

  unsigned short uiVFD2Offset;    /* in bits                      */
  unsigned char ucVFD2Range;      /* in bits                      */
  unsigned char ucVFD2Pattern;    /* HVFD_ENABLED, HVFD_STATIC, */
  /* HVFD_INCR, HVFD_DEC, */
  /* HVFD_RANDOM, HVFD_NONE */
  unsigned long ulVFD2PatternCount;/* from 0(off) to 16,777,215 */
  /* number to incr. or decr. */
  /* through when using inc or */
  /* dec pattern */
  unsigned char ucVFD2StartVal[6];/* the initial VFD byte pattern*/

  unsigned short uiVFD3Offset;    /* in bytes                     */
  unsigned short uiVFD3Range;     /* in bytes                     */
  unsigned char ucVFD3Enable;     /* HVFD_ENABLED, HVFD_NONE      */

  unsigned char ucTagField;       /* 0 = off, 1 = insert signature */
  /* field into each frame */

  unsigned char DestinationMAC[6];/* the Stream's Dest MAC addr */
  unsigned char SourceMAC[6];     /* the Stream's Source MAC addr */
  unsigned char TypeOfService;    /* */
  unsigned char TimeToLive;       /* number of "hops" until frame */
  /* will be dropped */
  unsigned short InitialSequenceNumber; /* Initial sequence number*/
  unsigned char DestinationIP[4];/* Dest IP addr(e.g. 192.100.5.3) */
  unsigned char SourceIP[4];      /* Src IP addr(e.g. 192.100.5.4) */
  unsigned char Netmask[4];       /* Network Mask(e.g. 255.255.0.0)*/
  unsigned char Gateway[4];       /* Gateway addr(e.g. 192.100.1.1)*/
  unsigned short UDPSrc;          /* UDP Source Port */
  unsigned short UDPDest;         /* UDP Dest Port */
  unsigned short UDPLen;          /* UDP Length field */
  unsigned char extra[12];        /* reserved */
  unsigned short uiActualSequenceNumber;/* Actual Sequence number */
  unsigned long ulARPStart;       /* Return value for the Time of */
  /* the last ARP initiated */
  unsigned long ulARPEnd;         /* Return value for the Time of */
  /* the last ARP completed */
  unsigned long ulARPGap;         /* The Time between ARPs */
} StreamUDP;
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_DUP_PVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Duplicate an existing PVC config</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_DUP_PVC,</td>
</tr>
<tr>
<td></td>
<td>&lt;index&gt;, &lt;count&gt;, 0,</td>
</tr>
<tr>
<td></td>
<td>(void*)pFRPvcTableEntry,</td>
</tr>
<tr>
<td></td>
<td>sizeof(FRPvcTableEntry),</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
</tbody>
</table>

**Related Structure** FRPvcTableEntry

Since the WAN firmware does not allow the deletion of a single PVC, a new PVC configuration setup requires a FR_PVC_DELETE_ALL command to remove all old PVCs. All test traffic is automatically stopped. Then PVC configuration command is used to add/modify a PVC configuration. If a PVC is modified, test traffic is immediately stopped. However, adding a PVC does not stop test traffic. In other words, the PVC won't be used until test traffic is started next time after a COMMIT CONFIGURATION command. When a PVC is added/modified, it will appear as a new and inactive PVC when UNI mode is DCE, as an active PVC when UNI is disabled and as a configured PVC when UNI mode is DTE. The link status will not be affected.

If ulDLCI exists, the PVC is overwritten with the new configuration.

typedef struct tagFRPvcTableEntry
{
  unsigned long ulCIR;        /* Committed Info. Rate in bps */
  unsigned short uiDLCI;       /* Local DLCI number: 1-1022 */
  unsigned short uiFrameSize;  /* OBSOLETE */
  unsigned long ulBc;         /* reserved - Committed burst size in kbits */
  unsigned long ulAccessRate; /* reserved - Physical Access Rate. NA for F/R */
  unsigned long ulBe;         /* reserved - Excess Burst size in kbits */
  unsigned short uiFrameRate; /* Frame rate per seconds */
  unsigned short uiStreamCount;/* OBSOLETE: No. configured streams */
  unsigned char ucReserved[12];
} FRPvcTableEntry; /* FR_PVC_CFG_TBL_ENT_T; */

**Comment**
## iTYPE1

<table>
<thead>
<tr>
<th>Description</th>
<th>FR_DUP_STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>iType1</strong></td>
<td><strong>FR_DUP_STREAM</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>reserved - duplicate streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FR_DUP_STREAM, &lt;index&gt;, &lt;count&gt;, 0, (void*)pStreamSmartBits, sizeof(StreamSmartBits), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

### Related Structure
StreamSmartBits

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern.

Then define the ProtocolHeader array. This array (of up-to 64 bytes) is used similarly to VFD1. It overwrites the background pattern at the specified offset and range. (ucVFD1Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits {
    unsigned char ucActive; /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength; /* Reserved */
    unsigned short uiRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC */
    /* HVFD_INCR, HVFD_DECWR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern*/
    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC */
    /* HVFD_INCR, HVFD_DECWR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern*/
    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes; Number of elements */
    /* to use from ProtocolHeader */
    /* No elements are used beyond */
    /* the single specified range. */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */
    unsigned char ucTagField; /* 0 = off, 1 = insert signature*/
    /* field into each frame */
    unsigned char ProtocolHeader[64]; /* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_FILL_PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Config global background fill pattern</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_FILL_PATTERN, 0, 0, 0, (void*)pUChar, sizeof(UChar), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_HIST_LATENCY_DISTRIBUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_HIST_LATENCY_DISTRIBUTION, 0, 0, 0, (void*)pLayer3HistDistribution, sizeof(Layer3HistDistribution), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_HIST_SEQUENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_HIST_SEQUENCE, 0, 0, NULL, 0, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_HIST_V2_LATENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_HIST_V2_LATENCY, 0, 0, 0, (void*)pLayer3HistLatency, sizeof(Layer3HistLatency), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>iType1</td>
<td>FR_HIST_V2_LATENCY_PER_STREAM</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>int HTSetStructure(FR_HIST_V2_LATENCY_PER_STREAM, 0, 0, 0, (void*)Layer3V2HistDistribution, sizeof(Layer3V2HistDistribution), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>No Related Structure</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_IP_SUBNET_DEREG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Remove a new or existing IP subnet addr</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_IP_SUBNET_DEREG, 0, 0, 0, (void*)pFRIPSubnetDeRegister, sizeof(FRIPSubnetDeRegister), iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Related Structure</td>
<td>FRIPSubnetDeRegister</td>
</tr>
</tbody>
</table>

This structure is used to register and de-register IP subnets for the line. One or more PVC can belong to the same IP subnet. So, up to FR_MAX_IPSUBNET_ID IP subnets can be registered for each WAN card, with one PVC for each IP subnet. In each register structure, one or more IP subnets can be defined, with the first short word indicating the number of IP subnets to follow. In each de-register structure, the number of IP subnets to de-register and the starting IP subnet ID must be provided. The IP subnet ID ranges from 0 to FR_MAX_IPSUBNET_ID.

Example:

To register two IP subnet addresses, 192.100.1.1 and 192.100.2.1 with netmask of 255.255.255.0 and default gateway of 192.100.2.2

```
byte      content
 0         00
 1         02
 2         192    beginning of first IP subnet
 3         100
 4         001
 5         001
 6         255
 7         255
 8         255
 9         000
10        192
11        100
12        002
13        002
14        192    beginning of second IP subnet
15        100
16        002
17        001
18        255
19        255
20        255
21        000
22        192
23        100
24        002
25        002
```

typedef struct tagFRIPSubnetDeRegister
{
    unsigned short   uiSubnetCount; /* No. of subnets to delete */
    unsigned short   uiIPSubnetId;  /* Starting IP subnet ID */
} FRIPSubnetDeRegister;

SmartLib - Structure Reference 163
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_IP_SUBNET_REG</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Config a new or existing IP subnet addr</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FR_IP_SUBNET_REG, 0, 0, 0, (void*)pFRIPSubnetRegister, sizeof(FRIPSubnetRegister), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>FRIPSubnetRegister</td>
</tr>
</tbody>
</table>

```c
typedef struct tagFRIPSubnetRegister
{
  unsigned short  uiIPSubnetId;  /* ID starting as 0 */
  unsigned char   ucIPAddress[4];  /* Local IP addr for this subnet */
  unsigned char   ucNetmask[4];   /* Local network mask */
  unsigned char   ucDefaultGateway[4];  /* Default gateway */
} FRIPSubnetRegister;
```
<table>
<thead>
<tr>
<th>iTyple1</th>
<th>FR_LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Config physical layer</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FR_LINE, 0, 0, 0, (void*)pFRLineCfg, sizeof(FRLineCfg), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>FRLineCfg</td>
</tr>
</tbody>
</table>

Line configurations cause the line to be restarted. The new parameters are stored in non-volatile memory. The line is brought down for only a short duration (much less than 1 second). If the line is disabled, it will remain so.

NOTE: Currently, an interface with physical DTE interface MUST use external clocking whereas a DCE MUST use internal clocking. This limitation will be removed in the near future.

```c
typedef struct tagFRLineCfg
{
    unsigned long ulSpeed;    /* Line speed in bps as defined by FR_LINE_SPEED_XXX above */
    unsigned long ulProgBits; /* reserved - MUST BE SET TO 0 */
    /* The Program Bit Pattern (21-25 bits) from */
    /* BitCalc utility for programming the ICD-2053 */
    /* Clock Chip driving the HDLC controller for */
    /* line speeds of 56K-8Mega Bits Per Sec. */
    unsigned long ulProgBitsLen; /* reserved - MUST BE SET TO 0 */
    /* Length of the above program bit pattern */
    /* BitCalc utility */
    unsigned char ucLineMode; /* line physical mode */
    /* FR_CARD_DTE or FR_CARD_DCE */
    unsigned char ucClocking; /* clocking type */
    /* FR_CARD_CLK_EXTERNAL or FR_CARD_CLK_INTERNAL */
    unsigned char ucClkPolarity; /* clock edge for tx data */
    /* FR_CLK_RISING_EDGE or FR_CLK_FALLING_EDGE */
    unsigned char ucEncoding; /* tx/rx data encoding */
    /* FR_NRZ_ENCODE or FR_NRZI_ENCODE */
    unsigned char ucGapCtl; /* min. # of interFrame flags */
    /* MIN. GAP VALUE or greater */
    unsigned char ucLoopbackOn; /* 1 = on or 0 off */
    unsigned char ucCrcOff; /* 1 = CRC off */
    unsigned char ucUseCRC32; /* 1 = CRC32, 0 = CRC16 */
    unsigned char ucDataUnchanged; /* 1 = no zero insertion, 0 = zero insertion */
    /* set indicate to turn on/detect signal */
    unsigned char ucDsrOn; /* DCE */
    unsigned char ucCtsOn; /* DCE */
    unsigned char ucDcdOn; /* DCE */
    unsigned char ucTmoOn; /* DCE */
    unsigned char ucDTrOn; /* DTE */
    unsigned char ucRtsOn; /* DTE */
    unsigned char ucRdiOn; /* DTE : reserved */
    unsigned char ucLbOn; /* DTE : reserved */
    unsigned char ucRxClkPolarity; /* clock edge for tx data */
    /* if uclineMode,above, is set to FR_CARD_DCE */
    unsigned char ucRxClkPolarity; /* clock edge for tx data */
    unsigned char ucReserved[13];
} FRLineCfg; /* FR_LINE_CFG_PARAM_T */
```
### iType1 | FR_LMI
---|---
**Description** | Config LMP signaling timers, counter and mode

**Usage**
```c
int HTSetStructure(FR_LMI,
                   0, 0, 0,
                   (void*)pFRLmiCfg,
                   sizeof(FRLmiCfg),
                   iHub, iSlot, iPort);
```

**Related Structure** | FRLmiCfg
---|---

```c
typedef struct tagFRLmiCfg
{
  unsigned char ucLinkManagement; /* LMP protocol version */
  unsigned char ucUNIMode;        /* UNI mode */
  unsigned char ucNN1;    /* n391/nN1 : DTE Full Polling Cycle */
  unsigned char ucNN2;    /* NA */
  unsigned char ucNN3;    /* n393/nN3 : Monitor events count */
  unsigned char ucNN4;    /* NA */
  unsigned char ucNT1;    /* T391/nT1 : DTE Link Integrity Verification Timer (secs) */
  unsigned char ucNT2;    /* T392/nT2: DCE Polling Verification Timer (secs) */
  unsigned char ucNT3;    /* NA */
} FRLmiCfg;   /* FR_LMI_CFG_PARAMS_T; */
```

**Comment**

---

### iType1 | FR_MOD_UDP_STREAM
---|---
**Description** | reserved

**Usage**
```c
int HTSetStructure(FR_MOD_UDP_STREAM,
                   <index>, <count>, 0,
                   (void*)pStreamUDP,
                   sizeof(StreamUDP),
                   iHub, iSlot, iPort);
```

**Related Structure** | StreamUDP
---|---

This structure is used to define UDP compliant Streams.

The IP checksums are automatically calculated using the supplied field value and inserted into the frame.

The UDP checksum is set to 0, and so is not calculated.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).

NOTE: The Signature field is 18 bytes long, and laid into the frame just before the CRC. Use caution when defining the frame length so that the Signature does not overwrite important data.
typedef struct tagStreamUDP
{
    unsigned char ucActive;        /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType;  /* use STREAM_PROTOCOL_UDP */
    unsigned char ucRandomLength; /* Reserved */

    unsigned char ucRandomData;    /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength;  /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */

    /***************************************************************************/
    /*                                                                       */
    /*  For ETHERNET, cards VFD1, VFD2, and VFD3 structure members */
    /* are reserved for later use. Set to 0. */
    /***************************************************************************/
    unsigned short uiVFD1Offset;   /* in bits */
    unsigned char ucVFD1Range;     /* in bits */
    unsigned char ucVFD1Pattern;   /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6];  /* the initial VFD byte pattern*/

    unsigned short uiVFD2Offset;   /* in bits */
    unsigned char ucVFD2Range;     /* in bits */
    unsigned char ucVFD2Pattern;   /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6];  /* the initial VFD byte pattern*/

    unsigned short uiVFD3Offset;   /* in bytes */
    unsigned short uiVFD3Range;    /* in bytes */
    unsigned char ucVFD3Enable;    /* HVFD_ENABLED, HVFD_NONE */
    /***************************************************************************/
    unsigned char ucTagField;      /* 0 = off, 1 = insert signature */
    /* field into each frame */
    unsigned char DestinationMAC[6];  /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6];    /* the Stream's Source MAC addr */
    unsigned char TypeOfService;  /* */
    unsigned char TimeToLive;      /* number of "hops" until frame */
    /* will be dropped */
    unsigned short InitialSequenceNumber; /* Initial sequence number*/
    unsigned char DestinationIP[4]; /* Dest IP addr (e.g. 192.100.5.3) */
    unsigned char SourceIP[4];   /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4];    /* Network Mask (e.g. 255.255.0.0) */
    unsigned char Gateway[4];    /* Gateway addr (e.g. 192.100.1.1) */
    unsigned short UDPSrc;        /* UDP Source Port */
    unsigned short UDPDest;       /* UDP Dest Port */
    unsigned short UDPLen;        /* UDP Length field */
    unsigned char extra[12];      /* reserved */
    unsigned short uiActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPStart;      /* Return value for the Time of */
    /* the last ARP initiated */
    unsigned long ulARPEnd;        /* Return value for the Time of */
    /* the last ARP completed */
    unsigned long ulARPGap;        /* The Time between ARPs */
} StreamUDP;

Comment
For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.

**iType1** | **FR_PVC**
---|---
**Description** | Config a new or existing PVC
**Usage** | int HTSetStructure(FR_PVC,
0, 0, 0, (void*)pFRPvcTableEntry,
sizeof(FRPvcTableEntry),
iHub, iSlot, iPort);

**Related Structure** | FRPvcTableEntry

Since the WAN firmware does not allow the deletion of a single PVC, a new PVC configuration setup requires a FR_PVC_DELETE_ALL command to remove all old PVCs. All test traffic is automatically stopped. Then PVC configuration command is used to add/modify a PVC configuration. If a PVC is modified, test traffic is immediately stopped. However, adding a PVC does not stop test traffic. In other words, the PVC won’t be used until test traffic is started next time after a COMMIT CONFIGURATION command. When a PVC is added/modified, it will appear as a new and inactive PVC when UNI mode is DCE, as an active PVC when UNI is disabled and as a configured PVC when UNI mode is DTE. The link status will not be affected.

If ulDLCI exists, the PVC is overwritten with the new configuration.

typedef struct tagFRPvcTableEntry
{
    unsigned long ulCIR;        /* Committed Info. Rate in bps */
    unsigned short uiDLCI;       /* Local DLCI number: 1-1022 */
    unsigned short uiFrameSize;  /* OBSOLETE */
    unsigned long ulBc;         /* reserved - Committed burst size in kbits */
    unsigned long ulAccessRate; /* reserved - Physical Access Rate. NA for F/R */
    unsigned long ulBe;         /* reserved - Excess Burst size in kbits */
    unsigned short uiFrameRate; /* Frame rate per seconds */
    unsigned short uiStreamCount; /* OBSOLETE: No. configured streams */
    unsigned long ulIPSubnetId; /* IP subnet ID: If not defined, set to FR_IPSUBNET_NOT_DEFINED otherwise set to 0 - FR_MAX_IPSUBNET_ID */
    unsigned char ucReserved[12];
} FRPvcTableEntry; /* FR_PVC_CFG_TBL_ENT_T */

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_PVC_CTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Control a single PVC : disable/enable</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetStructure(FR_PVC_CTRL, 0, 0, 0, (void*)pFRPvcControl, sizeof(FRPvcControl), iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>FRPvcControl</td>
</tr>
</tbody>
</table>

```c
typedef struct tagFRPvcControl {
    unsigned short  uiDLCI;     /* PVC dlci number */
    unsigned char   ucEnable;   /* 1 = enable; 0 = disable */
    unsigned char   ucReserved[3];
    unsigned long   ulReserved[4];
} FRPvcControl;  /* FR_PVC_CTL_T; */
```

<p>| <strong>Comment</strong> | /* FR_PVC_CTL_T */ |</p>
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_PVC_STREAM_MAP_CFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Config per stream PVC related params</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetStructure(FR_PVC_STREAM_MAP_CFG, 0, 0, 0, (void*)pFRPvcStrmMapCfg, sizeof(FRPvcStrmMapCfg), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**  
FRPvcStrmMapCfg

> Since the WAN firmware does not allow the deletion of a single stream, a new stream configuration setup requires a FR_STREAM_DELETE_ALL command to remove all existing streams. All test traffic is automatically stopped. Then Stream Configuration command is used to add/modify a stream configuration. Test traffic will continue. However, some of the stream parameters like protocol type change will not go into effect until test traffic is restarted again. Note that the new stream will not be included in test traffic until test traffic is restarted.

After the usual stream configuration is done. More stream parameters related to WAN is configured for each stream using the FRPvcStreamMapCfg structure. It is used to determine length of global background fill to use, FCS and ABORT error insertion, to set FECN, BECN, DE and CR bit in the Q.922 header for frame relay, and also the RFC1490 encap type. Note also it is not recommended to mix bridging and routing encap on the same PVC.

**Either use All Bridging or All Routing Format. Raw Encap Can Always be Combined With Bridgeing and Routing.**

A streamID must exist when a mapping structure is used. Otherwise it has no effect. A uiStreamID of FR_STREAMID_NOT_DEFINED must be used if a new stream configuration is used. However, uiStreamID MUST be set to the proper ID.

```c
typedef struct tagFRPvcStrmMapCfg
{
    unsigned long ulStreamId;
    unsigned short uiDLCI;
    unsigned short uiVfdState; /* reserved */
    unsigned short uiBgFillLen; /* number of bytes to copy to frame from background field */
    unsigned short uiMinFrameSize; /* reserved - if Random Frame size */
    unsigned short uiMaxFrameSize; /* reserved - is enabled for stream */
    unsigned char ucFcsError; /* 1 = cause FCS error */
    unsigned char ucAbortFlag; /* 1 = gen abort flag following frame */
    unsigned char ucEncapType; /* RFC encap. type for this PVC */
    unsigned char ucCR; /* 1 = set CR bit on all frames in stream */
    unsigned char ucFECN; /* 1 = set FECN bit on all frames in stream */
    unsigned char ucDE; /* 1 = set DE bit on all frames in stream */
    unsigned char ucReserved1;
    unsigned char ucEncapHeader[MAX_ENCAPHEADER_LEN]; /* RFC1490 encap */
    // /* reserved -Starting from xfr header + RFC1490 */
    unsigned char ucReserved[12];
} FRPvcStrmMapCfg; /* FR_PVC_STREAM_MAP_CFG_T; */
```

**Comment**
**iType1**  
**FR_STRM_CTRL**

**Description**  
Control a stream: disable/enable, generate errs etc

**Usage**  
int HTSetStructure(FR_STRM_CTRL,  
0, 0, 0,  
(void*)pFRStreamControl,  
ssizeof(FRStreamControl),  
iHub, iSlot, iPort);  

**Related Structure**  
FRStreamControl

typedef struct tagFRStreamControl  
{  
unsigned long ulStreamId; /* stream ID */  
unsigned char ucEnable; /* 1 = enable; 0 = disable */  
unsigned char ucFcsErr; /* 1 = cause FCS error */  
unsigned char ucAbortFlag; /* 1 = generate abort flag following frame */  
unsigned char ucReserved;  
unsigned long ulReserved[4];  
} FRStreamControl; /* FR_STREAM_CTL_T */

**Comment**

---

**iType1**  
**FR_T1E1_LINE**

**Description**  
Config T1E1 physical layer

**Usage**  
int HTSetStructure(FR_T1E1_LINE,  
0, 0, 0,  
(void*)pFRT1E1LineCfg,  
ssizeof(FRT1E1LineCfg),  
iHub, iSlot, iPort);  

**Related Structure**  
FRT1E1LineCfg

typedef struct tagFRT1E1LineCfg  
{  
unsigned char ucLineMode; /* line physical mode */  
unsigned char ucClocking; /* Tx clocking type */  
unsigned char ucDataEncoding; /* NRZ v.s. NRZI */  
unsigned char ucGapCtl; /* # of interFrame flags */  
unsigned char ucCrcOff; /* boolean - TRUE : CRC off */  
unsigned char ucUseCRC32; /* boolean - TRUE : CRC32 */  
unsigned char ucDataUnchanged; /* boolean */  
/* Set Indicates Rcv DataIn Unchanged  
(No Zero deletion or Crc check) */  
/* Line interface control flags */  
unsigned char ucLoopbackEnable; /* Loopback mode */  
unsigned char ucLineBuildout; /* Pulse shaping for PHY */  
unsigned char ucLineCoding; /* Line symbol coding */  
unsigned char ucLineFraming; /* Framing format for PHY */  
unsigned char ucChannels[32]; /* T1-24, E1-32 channels, 0: not selected */  
/*  
* The following fields are reserved and should  
* be set to 0 in any download.  
*/  
unsigned char ucReserved[9];  
} FRT1E1LineCfg;

**Comment**
### FR_TRIGGER

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_TRIGGER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Config global tx and receive triggers</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTSetStructure(FR_TRIGGER, 0, 0, 0, (void*)pFRTriggerCfg, sizeof(FRTriggerCfg), iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

### Related Structure
**FRTriggerCfg**

```c
typedef struct tagFRTriggerCfg
{
    unsigned char ucEnable;    /* Enable/disable triggers */
    unsigned char ucDirection; /* Tx or Rx - currently used for both*/
    unsigned char ucCompCombo; /* Trigger types: one of FR_TRIG_COMP*/
    unsigned char ucReserved1;

    unsigned short uiTrig1Offset;     /* Offset into frame for pattern test in bytes */
    unsigned short uiTrig1Range;      /* Number of bytes to match */
    unsigned char ucTrig1Pattern[6]; /* Pattern to match */
    unsigned char ucTrig1Mask[6];     /* Bit mask for pattern */

    unsigned short uiTrig2Offset;     /* Offset into frame for pattern test in bytes */
    unsigned short uiTrig2Range;      /* Number of bytes to match */
    unsigned char ucTrig2Pattern[6]; /* Pattern to match */
    unsigned char ucTrig2Mask[6];     /* Bit mask for pattern */

    unsigned char ucReserved[20];
} FRTriggerCfg; /* FR_TRIG_PARAM_T; */
```

### Comment

### FR - HTGetStructure

#### iType1 FR_AGGR_LATENCY_DISTRIBUTION_INFO

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_AGGR_LATENCY_DISTRIBUTION_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Full Latency Distribution histogram results</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(FR_AGGR_LATENCY_DISTRIBUTION_INFO, 0, 0, 0, (void*)pLayer3StreamDistributionInfo, sizeof(Layer3StreamDistributionInfo), iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

No Related Structure

### Comment

### iType1 FR_AGGR_SEQUENCE_INFO

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_AGGR_SEQUENCE_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get Sequence Tracking histogram results</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(FR_AGGR_SEQUENCE_INFO, 0, 0, 0, (void*)pLayer3SequenceInfo, sizeof(Layer3SequenceInfo), iHub, iSlot, iPort);</code></td>
</tr>
</tbody>
</table>

No Related Structure

### Comment
### FR_AGGR_V2_LATENCY_INFO

**Description**
Get 32-bit Latency over Time histogram results

**Usage**
```c
int HTGetStructure(FR_AGGR_V2_LATENCY_INFO,
                   0, 0, 0,
                   (void*)pLayer3LongLatencyInfo,
                   sizeof(Layer3LongLatencyInfo),
                   iHub, iSlot, iPort);
```

**Comment**
No Related Structure

---

### FR_AGGR_V2_LATENCY_PER_STREAM_INFO

**Description**
Get 32-bit Latency Distribution and Per Stream histogram results

**Usage**
```c
int HTGetStructure(FR_AGGR_V2_LATENCY_PER_STREAM_INFO,
                   0, 0, 0,
                   (void*)pLayer3StreamLongLatencyInfo,
                   sizeof(Layer3StreamLongLatencyInfo),
                   iHub, iSlot, iPort);
```

**Comment**
No Related Structure

---

### FR_CARD_VERSION_INFO

**Description**
Get card firmware versions

**Usage**
```c
int HTGetStructure(FR_CARD_VERSION_INFO,
                   0, 0, 0,
                   (void*)pFRVersionInfo,
                   sizeof(FRVersionInfo),
                   iHub, iSlot, iPort);
```

**Related Structure**
FRVersionInfo

This structure will return the revision information associated with the firmware, and diagnostic and identification information associated with the hardware. The format of the version information for the main firmware version is as follows:

- Bit 15: TBD - General release flag: Set: Released, Unset: Beta
- Bits 0-14: Versioning information
  - Major release: (ushort/100) % 10
  - Minor release: ushort % 100
  - Build: (ushort/1000) & 0x1F;

```c
typedef struct tagFRVersionInfo
{
    unsigned short    uiMainFwVersion; /* firmware version number */
    unsigned short    uiBootFwVersion; /* reserved */
    unsigned short    uiFpgaVersion; /* reserved */
} FRVersionInfo; /* FR_WANS_IDENTS_T */
```

**Comment**
### FR_DEFINED_STREAM_COUNT_INFO

**Description**

**Usage**

```c
int HTGetStructure(FR_DEFINED_STREAM_COUNT_INFO, 0, 0, 0, (void*)pULong, sizeof(ULong), iHub, iSlot, iPort);
```

No Related Structure

**Comment**

---

### FR_HIST_ACTIVE_TEST_INFO

**Description**

**Usage**

```c
int HTGetStructure(FR_HIST_ACTIVE_TEST_INFO, 0, 0, 0, (void*)pLayer3HistActiveTest, sizeof(Layer3HistActiveTest), iHub, iSlot, iPort);
```

No Related Structure

**Comment**

---

### FR_HIST_LATENCY_DISTRIBUTION_INFO

**Description**

**Usage**

```c
int HTGetStructure(FR_HIST_LATENCY_DISTRIBUTION_INFO, <index>, <count>, 0, (void*)pLayer3StreamDistributionInfo, sizeof(Layer3StreamDistributionInfo), iHub, iSlot, iPort);
```

No Related Structure

**Comment**

---

### FR_HIST_SEQUENCE_INFO

**Description**

**Usage**

```c
int HTGetStructure(FR_HIST_SEQUENCE_INFO, <index>, <count>, 0, (void*)pLayer3SequenceInfo, sizeof(Layer3SequenceInfo), iHub, iSlot, iPort);
```

No Related Structure

**Comment**
### FR_HIST_TYPE_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th>Get histogram result data</th>
</tr>
</thead>
</table>

**Usage**

```c
int HTGetStructure(FR_HIST_TYPE_INFO,
                   0, 0, 0,
                   NULL,
                   0,
                   iHub, iSlot, iPort);
```

### FR_HIST_V2_LATENCY_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
</table>

**Usage**

```c
int HTGetStructure(FR_HIST_V2_LATENCY_INFO,
                   <index>, <count>, 0,
                   (void*)pLayer3LongLatencyInfo,
                   sizeof(Layer3LongLatencyInfo),
                   iHub, iSlot, iPort);
```

### FR_HIST_V2_LATENCY_PER_STREAM_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
</table>

**Usage**

```c
int HTGetStructure(FR_HIST_V2_LATENCY_PER_STREAM_INFO,
                   <index>, <count>, 0,
                   (void*)pLayer3StreamLongLatencyInfo,
                   sizeof(Layer3StreamLongLatencyInfo),
                   iHub, iSlot, iPort);
```

### FR_IP_STREAM_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th>Get IP stream config info.</th>
</tr>
</thead>
</table>

**Usage**

```c
int HTGetStructure(FR_IP_STREAM_INFO,
                   <index>, 0, 0,
                   (void*)pStreamIP,
                   sizeof(StreamIP),
                   iHub, iSlot, iPort);
```

**Related Structure**

StreamIP

This structure is used to define IP compliant Streams.

The IP checksum is automatically calculated using the supplied header fields and inserted into the IP header.

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamIP
{
    unsigned char ucActive; /* 1 = Enable Stream, 0 = Disable Stream */
    unsigned char ucProtocolType; /* use STREAM_PROTOCOL_IP */
    unsigned char ucRandomLength; /* Reserved */
    unsigned char ucRandomData; /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength; /* frame length not counting CRC */
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */

    /********************************************************************
    * For ETHERNET, cards VFD1, VFD2, and VFD3 structure members *
    * are reserved for later use. Set to 0.                          *
    *----------------------------------------------------------------------*/

    unsigned short uiVFD1Offset; /* in bits */
    unsigned char ucVFD1Range; /* in bits */
    unsigned char ucVFD1Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFDINCREMENT, HVFDDECREMENT, */
    /* HVFDRANDOM, HVFDNONE */
    unsigned long ulVFD1PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6]; /* the initial VFD byte pattern */

    unsigned short uiVFD2Offset; /* in bits */
    unsigned char ucVFD2Range; /* in bits */
    unsigned char ucVFD2Pattern; /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFDINCREMENT, HVFDDECREMENT, */
    /* HVFDRANDOM, HVFDNONE */
    unsigned long ulVFD2PatternCount; /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6]; /* the initial VFD byte pattern */

    unsigned short uiVFD3Offset; /* in bytes */
    unsigned short uiVFD3Range; /* in bytes */
    unsigned char ucVFD3Enable; /* HVFD_ENABLED, HVFD_NONE */

    /********************************************************************
    unsigned char ucTagField; /* 0 = off, 1 = insert Signature */
    /* field into each frame */
    unsigned char DestinationMAC[6]; /* the Stream's Dest MAC addr */
    unsigned char SourceMAC[6]; /* the Stream's Source MAC addr */
    unsigned char TypeOfService; /* */
    unsigned char TimeToLive; /* number of "hops" until frame */
    /* will be dropped */
    unsigned short InitialSequenceNumber; /* Initial sequence number */
    unsigned char DestinationIP[4]; /* Dest IP addr (e.g. 192.100.5.1) */
    unsigned char SourceIP[4]; /* Src IP addr (e.g. 192.100.5.4) */
    unsigned char Netmask[4]; /* Network Mask (e.g. 255.255.0.0) */
    unsigned char Gateway[4]; /* Gateway addr (e.g. 192.100.1.1) */
    unsigned char Protocol; /* 4=IP on the IP assigned list */
    unsigned char extra[17]; /* reserved */
    unsigned long uiActualSequenceNumber; /* Actual Sequence number */
    unsigned long ulARPStart; /* Return value for the Time of */
    /* the last ARP initiated */
    unsigned long ulARPEnd; /* Return value for the Time of */
    /* the last ARP completed */
    unsigned long ulARPGap; /* The Time between ARPs */
    } StreamIP;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
<table>
<thead>
<tr>
<th>iTyp1</th>
<th>FR_LINK_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Get link statistics counters</td>
</tr>
</tbody>
</table>
| Usage    | int HTGetStructure(FR_LINK_INFO,  
|          | 0, 0, 0,  
|          | (void*)pFRLinkInfo,  
|          | sizeof(FRLinkInfo),  
|          | iHub, iSlot, iPort) ; |

**Related Structure**  
FRLinkInfo

Counter values reflects counts since last cleared.

Link statistics are returned as in the link statistics structure. All ARP, INARP, PING, RIP and SNMP frames are considered as stack frames for transmission purpose. All frames with errors are counted in the receive byte count, but not in the receive frame count. LMI is not counted in the link statistics. However, all ARP, RIP, SNMP and PING are counted in link receive statistics. So ARP and PING are counted on the link receive statistics and separately in their own counters. Furthermore, DE bit, FECN and BECN are counted on top of the regular statistics.
typedef struct  tagFRLinkInfo
{
    unsigned long  ulTimestamp;      /* timestamp of last rate update */
    unsigned long  ulRxFrameRate;    /* Rate Rx frames */
    unsigned long  ulRxByteRate;     /* Rate Rx bytes */
    unsigned long  ulRxFcsErrRate;   /* Rate Rx frames with FCS error */
    unsigned long  ulRxTriggerRate;  /* Rate Rx Trigger Match */
    unsigned long  ulRxAbortRate;    /* Rate Rx Abort frames */
    unsigned long  ulRxInvLenErrRate;/* Rate Rx Invalid Length frames */
    unsigned long  ulRxNonOctetAlignedErrRate; /* Rate of Non-octet alignment error frames */
    unsigned long  ulRxOverflowErrRate; /* Rate Rx overflow err frames */
    unsigned long  ulRxFrames;       /* Count Rx frames */
    unsigned long  ulRxBytes;        /* Count Rx bytes */
    unsigned long  ulRxFcs_err;      /* Count Rx FCS errors */
    unsigned long  ulRxTrigger;      /* Count Rx Trigger Match Count */
    unsigned long  ulRxAbort;        /* Count Rx Invalid length frames */
    unsigned long  ulRxInvLenErr;    /* Count Rx invalid length error */
    unsigned long  ulRxNonOctetAlignedErr; /* Count Rx non-oct alignment error frames */
    unsigned long  ulRxOverflowErr;  /* Count Rx overflow error */
    unsigned long  ulRxIdleSeq;      /* No. Rx Idle sequence */
    unsigned long  ulRxDeFrames;     /* No. Rx frames with DE bit set */
    unsigned long  ulRxBECNCount;    /* No. Rx frames with BECN bit set */
    unsigned long  ulRxFECNCount;    /* No. Rx frames with FECN bit set */
    unsigned long  ulRxInvalidPVC;   /* No. Rx frames with unconfigured PVC */
    unsigned long  ulRxTrigLatency;  /* First Rx Trig Match Time(100ns)*/
    unsigned long  ulRxTags;         /* Rx frames with Netcom signature tag */
    unsigned long  ulRxStack;        /* Rx frames destined for local IP addr or source IP addr. in stream */
    unsigned long  ulRxInvARPReq;    /* Rx Inverse ARP frame */
    unsigned long  ulRxARPReq;       /* RX ARP request frame */
    unsigned long  ulRxARPReply;     /* RX ARP reply frame */
    unsigned long  ulRxPingReq;      /* Rx ICMP PING command */
    unsigned long  ulRxPingReply;    /* Rx ICMP PING response */
    unsigned long  ulTxFrameRate;    /* Rate Tx frames */
    unsigned long  ulTxByteRate;     /* Rate Tx bytes */
    unsigned long  ulTxFcsErrRate;   /* Rate Tx FCS error */
    unsigned long  ulTxAbortRate;    /* Rate Tx abort error */
    unsigned long  ulTxTriggerRate;  /* Rate Tx trigger match */
    unsigned long  ulTxFrames;       /* Count Tx frames */
    unsigned long  ulTxBytes;        /* Count Tx bytes */
    unsigned long  ulTxFcsErr;       /* Count Tx FCS error */
    unsigned long  ulTxAbort;        /* Count Tx Abort */
    unsigned long  ulTxDeFrames;     /* Count Tx DE bit set */
    unsigned long  ulTxBECNFrames;   /* Count Tx BECN bit set */
    unsigned long  ulTxFECNFrames;   /* Count Tx FECN bit set */
    unsigned long  ulTxTrigLatency;  /* Trigger 1st Match Time(100ns)*/
    unsigned long  ulTxStack;        /* non-test frames originating from CARD for ICMP, PING, etc */
    unsigned long  ulTxInvARPReq;    /* Count Tx Inverse ARP request */
    unsigned long  ulTxARPReq;       /* Count Tx ARP request */
    unsigned long  ulTxARPReply;     /* Count Tx ARP reply */
    unsigned long  ulTxPingReq;      /* Count Tx PING (ICMP) request */
    unsigned long  ulTxPingReply;    /* Count Tx PING (ICMP) reply */
    unsigned long  ulReserved[20];
} FRLinkInfo;   /* FR_LINK_STATS_T; */
### iType1 FR_LINK_STATUS_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th>Get link status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>int HTGetStructure(FR_LINK_STATUS_INFO, 0, 0, 0, (void*)pFRLinkStatusInfo, sizeof(FRLinkStatusInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**: FRLinkStatusInfo

This structure is used to retrieve physical line info, such as external clock line speed.

```c
typedef struct tagFRLinkStatusInfo {
  unsigned long ulSpeed; /* actual line speed in kbps for both DTE and DCE */
  unsigned long ulReserved[19];
} FRLinkStatusInfo; /* FR_LINE_STATUS_T; */
```

### Comment

### iType1 FR_LMI_INFO

<table>
<thead>
<tr>
<th>Description</th>
<th>Get LMI statistics counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>int HTGetStructure(FR_LMI_INFO, 0, 0, 0, (void*)pFRLmiInfo, sizeof(FRlmiInfo), iHub, iSlot, iPort);</td>
</tr>
</tbody>
</table>

**Related Structure**: FRLmiInfo

```c
typedef struct tagFRLmiInfo {
  unsigned long ulConfiguredPvc; /* No. configured PVC */
  unsigned long ulActivePvc; /* No. active PVC */
  unsigned long ulInactivePvc; /* No. disable PVC */
  unsigned long ulDisabledPvc; /* No. disable PVC */
  unsigned long ulTxStatusReq; /* No. STATUS ENQUIRY sent */
  unsigned long ulTxStatusMsg; /* No. STATUS MESSAGE sent */
  unsigned long ulTxFullStatusReq; /* No. FULL STATUS ENQUIRY sent */
  unsigned long ulTxFullStatusMsg; /* No. FULL STATUS MESSAGE sent */
  unsigned long ulTxStatusUpdate; /* reserved */
  unsigned long ulRxStatusReq; /* No. STATUS ENQUIRY received */
  unsigned long ulRxStatusMsg; /* No. STATUS MESSAGE received */
  unsigned long ulRxFullStatusReq; /* No. FULL STATUS ENQUIRY Rcvd */
  unsigned long ulRxFullStatusMsg; /* No. FULL STATUS MESSAGE Rcvd */
  unsigned long ulRxStatusUpdate; /* No. STATUS UPDATE MESSAGE Rcvd */
  unsigned long ulPvcCongestion; /* reserved */
  unsigned long ulNewPvc; /* No. new PVCs */
  unsigned long ulPvcDeleted; /* No. deleted PVCs */
  unsigned long ulPvcDeactivated; /* No. timers PVC deactivated */
  unsigned long ulMulticastIe; /* reserved */
  unsigned long ulInvalidFrame; /* No. invalid LMP frames Rcvd */
  unsigned long ulReserved[3];
} FRLmiInfo; /* FR_LMI_STATS_T; */
```

**Comment**
### FR_PVC_INFO

**Description**
Get per-PVC statistics counters

**Usage**
```c
int HTGetStructure(FR_PVC_INFO, <index>, 0, 0, (void*)pFRPvcMainInfo, sizeof(FRPvcMainInfo), iHub, iSlot, iPort);
```

**Related Structure**
FRPvcMainInfo

```c
typedef struct tagFRPvcMainInfo
{
    unsigned short  uiPadding;
    unsigned short  uiDLCI;

    unsigned long  ulTxFrameRate;    /* Rate Tx frames */
    unsigned long  ulTxByteRate;     /* Rate Tx bytes */
    unsigned long  ulTxFrames;       /* Total Tx frames */
    unsigned long  ulTxBytes;        /* Total Tx bytes */
    unsigned long  ulTxDeFrames;     /* Tx frames with DE bit set */
    unsigned long  ulTotFECNsSent;   /* Tx frames with FECN bit set */
    unsigned long  ulTotBECNsSent;   /* Tx frames with BECN bit set */
    unsigned long  ulTxFcsErr;       /* Tx Frames with FCS error */
    unsigned long  ulTxAbort;        /* Tx Frames set to abort */

    unsigned long  ulRxFrameRate;    /* Rate Rx frames */
    unsigned long  ulRxByteRate;     /* Rate Rx bytes */
    unsigned long  ulRxFrames;       /* Cumulative Total Rx frames */
    unsigned long  ulRxBytes;        /* Cumulative Total Rx bytes */
    unsigned long  ulRxDeFrames;     /* Rx frames with DE bit set */
    unsigned long  ulFECN;           /* Rx frames with FECN bit set */
    unsigned long  ulBECN;           /* Rx frames with BECN bit set */

    unsigned long  ulReserved[4];
} FRPvcMainInfo;    /* FR_PVC_MAIN_STATS_T */
```

**Comment**
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_PVC_STATUS_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get PVC status for all 1024 PVCs</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>int HTGetStructure(FR_PVC_STATUS_INFO, &lt;index&gt;, 0, 0, (void*)pFRPVCStatusInfo, sizeof(FRPVCStatusInfo), iHub, iSlot, iPort);</code></td>
</tr>
<tr>
<td><strong>Related Structure</strong></td>
<td>FRPVCStatusInfo</td>
</tr>
</tbody>
</table>

PVC status is returned in an array of bitmap starting with PVC 0 to PVC 1022 bit = 1 indicates ACTIVE, CONFIGURED or ENABLED. The application can request one, two or three groups of PVC status values in that order. The status values are stored in three separate contiguous bitmaps with length equal to one, two or three times `FR_PVC_STATUS_BITMAP_LEN`.

Example bit map of 128 bytes for DLCI 17 active/configured/enabled

```
*                BIT MAP
*                MSB       LSB
* byte 0         0 0 0 0 0 0 0 0
* byte 1         0 0 0 0 0 0 0 0
* byte 2         0 1 0 0 0 0 0 0      DLCI 17 active
* ...           *
* byte 127       0 0 0 0 0 0 0 0
* byte 128       0 0 0 0 0 0 0 0
* byte 129       0 0 0 0 0 0 0 0
* byte 130       0 1 0 0 0 0 0 0      DLCI 17 configured
* ...           *
* byte 255       0 0 0 0 0 0 0 0
* byte 256       0 0 0 0 0 0 0 0
* byte 257       0 0 0 0 0 0 0 0
* byte 258       0 1 0 0 0 0 0 0      DLCI 17 enabled
* ...           *
* byte 383       0 0 0 0 0 0 0 0
```

typedef struct tagFRPVCStatusInfo
{
  unsigned char ucBitMap[FR_MAX_STATUS_BITMAP_LEN];
  } FRPVCStatusInfo;

**Comment**
<table>
<thead>
<tr>
<th><strong>iType1</strong></th>
<th><strong>FR_SMARTBITS_STREAM_INFO</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Get raw stream config info.</td>
</tr>
</tbody>
</table>
| **Usage** | int HTGetStructure(FR_SMARTBITS_STREAM_INFO, 
<index>, 0, 0, 
(void*)pStreamSmartBits, 
sizeof(StreamSmartBits), 
iHub, iSlot, iPort) ; |
| **Related Structure** | StreamSmartBits |

This structure is used to create customized streams.

The Background Fill Pattern and the ProtocolHeader[] can be used in combination to construct highly customized frames.

Use HTFillPattern to specify the background pattern. Then define the ProtocolHeader array. This array (of up-to 64 bytes) is used similarly to VFD3. It overwrites the background pattern at the specified offset and range. (ucVFD3Enable must be set to HVFD_ENABLED).

A Netcom Systems Signature field, used for Histogram results, is inserted at the end of the payload if ucTagField is enabled (set to 1).
typedef struct tagStreamSmartBits {
    unsigned char ucActive;  /* 1=Enable Stream, 0=Disable Stream */
    unsigned char ucProtocolType;  /* use STREAM_PROTOCOL_SMARTBITS */
    unsigned char ucRandomLength;  /* Reserved */
    unsigned char ucRandomData;  /* 1 = Random Data, 0 = use the */
    /* cards background pattern */
    /* not available on Frame Relay */
    unsigned short uiFrameLength;  /* frame length not counting CRC*/
    /* 0 - 2048 for Ethernet */
    /* 0 - 8196 for Frame Relay */
}

typedef struct tagStreamSmartBits {
    unsigned short uiVFD1Offset;  /* in bits */
    unsigned char ucVFD1Range;  /* in bits */
    unsigned char ucVFD1Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD1PatternCount;  /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD1StartVal[6];  /* the initial VFD byte pattern*/
}

typedef struct tagStreamSmartBits {
    unsigned short uiVFD2Offset;  /* in bits */
    unsigned char ucVFD2Range;  /* in bits */
    unsigned char ucVFD2Pattern;  /* HVFD_ENABLED, HVFD_STATIC, */
    /* HVFD_INCR, HVFD_DECR, */
    /* HVFD_RANDOM, HVFD_NONE */
    unsigned long ulVFD2PatternCount;  /* from 0(off) to 16,777,215 */
    /* number to incr. or decr. */
    /* through when using inc or */
    /* dec pattern */
    unsigned char ucVFD2StartVal[6];  /* the initial VFD byte pattern*/
}

typedef struct tagStreamSmartBits {
    unsigned short uiVFD3Offset;  /* in bytes */
    unsigned short uiVFD3Range;  /* in bytes; Number of elements */
    /* to use from ProtocolHeader */
    /* No elements are used beyond */
    /* the single specified range. */
    unsigned char ucVFD3Enable;  /* HVFD_ENABLED, HVFD_NONE */
}

typedef struct tagStreamSmartBits {
    unsigned char ucTagField;  /* 0 = off, 1 = insert signature*/
    /* field into each frame */
    unsigned char ProtocolHeader[64];  /* Defines up to 64 bytes used as VFD3*/
} StreamSmartBits;

Comment

For related commands and instructions about working with Streams, see Chapter 1 of the Message Functions manual.
### FR_T1E1_LINE_INFO

**Description**  
Get T1E1 physical layer info

**Usage**  
```c
int HTGetStructure(FR_T1E1_LINE_INFO,
                   0, 0, 0,
                   (void*)pFRT1E1LineInfo,
                   sizeof(FRT1E1LineInfo),
                   iHub, iSlot, iPort);
```

**Related Structure**  
FRT1E1LineInfo

```c
typedef struct tagFRT1E1LineInfo
{
    unsigned short uiAlarmCurrent;
    unsigned short uiAlarmHistory;

    unsigned long ulCodeViolationC;
    unsigned long ulFrameErrorC;            /* SEF errors */
    unsigned long ulSyncErrorC;

    unsigned long ulCodeViolationR;
    unsigned long ulFrameErrorR;
    unsigned long ulSyncErrorR;
    unsigned long reserve[4];
} FRT1E1LineInfo;
```

### FR - HTSetCommand

**iType1**  
FR_CLEAR_COUNTERS_CMD

**Description**  
Clear all statistic counters

**Usage**  
```c
int HTSetCommand(FR_CLEAR_COUNTERS_CMD,
                 0, 0, 0,
                 NULL,
                 iHub, iSlot, iPort);
```

### FR_COMMIT_CFG

**iType1**  
FR_COMMIT_CFG

**Description**  
Commit PVC, stream and IP subnet config.

**Usage**  
```c
int HTSetCommand(FR_COMMIT_CFG,
                 0, 0, 0,
                 NULL,
                 iHub, iSlot, iPort);
```
<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_DISABLE_PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Disable WAN port</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_DISABLE_PORT, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>No Related Structure</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_ENABLE_PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Enable WAN port</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_ENABLE_PORT, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>No Related Structure</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_GROUP_MEMBER_CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Set card to be a member of group</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_GROUP_MEMBER_CMD, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>No Related Structure</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_GROUP_START_CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Start transmission if belong to group</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_GROUP_START_CMD, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>No Related Structure</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_GROUP_STEP_CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Send one frame if belong to group</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_GROUP_STEP_CMD, 0, 0, 0, NULL, iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>No Related Structure</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>iType1</td>
<td>FR_GROUP_STOP_CMD</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>Stop transmission if belong to group</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(FR_GROUP_STOP_CMD,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_NON_GROUP_CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>unset card from group</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(FR_NON_GROUP_CMD,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_PVC_DELETE_ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Delete all existing PVCs</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(FR_PVC_DELETE_ALL,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_SET_START_CFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Start PVC, stream and IP subnet addr. config</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(FR_SET_START_CFG,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_START_CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Start transmission</td>
</tr>
<tr>
<td>Usage</td>
<td>int HTSetCommand(FR_START_CMD,</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0,</td>
</tr>
<tr>
<td></td>
<td>NULL,</td>
</tr>
<tr>
<td></td>
<td>iHub, iSlot, iPort) ;</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>iType1</td>
<td>FR_STEP_CMD</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Sent one frame</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_STEP_CMD, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_STOP_CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Stop transmission</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_STOP_CMD, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iType1</th>
<th>FR_STREAM_DELETE_ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Delete all existing streams</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>int HTSetCommand(FR_STREAM_DELETE_ALL, 0, 0, 0, NULL, iHub, iSlot, iPort);</td>
</tr>
<tr>
<td><strong>No Related Structure</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td></td>
</tr>
</tbody>
</table>
Appendix A:
Obsolete and Removed Commands

This section contains a list of commands (iType1 parameters) that are not supported in SmartLib 3.05. At this time, these commands have been removed from the documentation only.

Implemented in Later Release
These commands were prematurely documented. They will appear in the documentation once their functions are fully implemented. They do still exist in the header files.

- L3_AGGR_LATENCY_DISTRIBUTION_INFO
- L3_AGGR_RAW_TAGS_INFO
- L3_AGGR_SEQUENCY_INFO
- L3_AGGR_V2_LATENCY_INFO
- L3_AGGR_V2_LATENCY_PER_STREAM_INFO

Obsolete
These commands were replaced with commands that support larger latency values. They are removed from the documentation only, and continue to function in SmartLib with no changes necessary to existing code.

<table>
<thead>
<tr>
<th>Command</th>
<th>Substitute (when released)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3_AGGR_LATENCY_INFO</td>
<td>(Substitute: L3_AGGR_V2_LATENCY_INFO)</td>
</tr>
<tr>
<td>L3_AGGR_LATENCY_PER_STREAM_INFO</td>
<td>(Substitute: L3_AGGR_V2_LATENCY_PER_STREAM_INFO)</td>
</tr>
<tr>
<td>L3_HIST_LATENCY_INFO</td>
<td>(Substitute: L3_HIST_V2_LATENCY_INFO.)</td>
</tr>
<tr>
<td>L3_HIST_LATENCY_PER_STREAM_INFO</td>
<td>(Substitute: L3_V2_LATENCY_PER_STREAM_INFO)</td>
</tr>
<tr>
<td>L3_HIST_LATENCY</td>
<td>(Substitute: L3_HIST_V2_LATENCY.)</td>
</tr>
<tr>
<td>L3_HIST_LATENCY_PER_STREAM</td>
<td>(Substitute: L3_V2_LATENCY_PER_STREAM, a combination histogram that includes Latency Per Stream.)</td>
</tr>
<tr>
<td>L3_HIST_LATENCY_PRECISION</td>
<td>(Not needed with new V2 Latency Histograms.)</td>
</tr>
</tbody>
</table>

Not Supported
These commands are not supported by the current Netcom Systems firmware. Prototype work was inadvertently included in the documentation. At this time, there are no plans to release the following commands. We regret any inconvenience this oversight might have caused you.

- FR_802_3_STREAM_INFO
- FR_ARP_STREAM_INFO
- FR_CAP_CNT_INFO
- FR_DEFINE_802_3_STREAM
- FR_DEFINE_ARP_STREAM
- FR_HIST_DATA_TYPE_INFO
- FR_HIST_LATENCY_INFO
FR_HIST_LATENCY_PER_STREAM_INFO
FR_HIST_SUMMARY_INFO
FR_HIST_TYPE_INFO
FR_INPUT_MSG_CMD
FR_RESET_CARD_CMD
FR_SET_RELEASE_CAP_CNT
FR_VFD3_BUFFER
L3_CAPTURE_SPECIAL_TYPE
L3_DEFINE_802_3_STREAM
L3_DEFINE_MULTI_802_3_STREAM
L3_DEFINE_MULTI_UDPDHCP_STREAM
L3_DEFINE_UDPDHCP_STREAM
L3_MOD_802_3_STREAM
L3_MOD_UDPDHCP_STREAM
L3_DHCP_EXT_INFO
L3_DHCP_INFO
L3_DHCP_STATE_INFO
L3_DHCP_STATS_INFO
L3_DHCP_FORCE_EXCHANGE_DONE
L3_DHCP_INITIATE_REQUEST_AND_DECLINE
L3_DHCP_REBOOT_CURRENT_LEASE
L3_DHCP_RELEASE_CURRENT_LEASE
L3_DHCP_RENEW_CURRENT_LEASE
L3_DHCP_START_DISCOVER