This manual, Version 5.1, corresponds to Standard Software Release 4.3.2.

TEKELEC
26580 Agoura Road
Calabasas, California
91302
Information in this documentation is subject to change without notice. Any software which is furnished in conjunction with or embedded within the product(s) described in this documentation is furnished under a license agreement and/or a nondisclosure agreement, and may be used only as expressly permitted by the terms of such agreements(s). Unauthorized use or copying of the software or this documentation can result in civil or criminal penalties.

No part of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, for any purpose without the express written permission of an authorized representative of Tekelec.

Copyright Tekelec 1991. All rights reserved.

Chameleon 32 and Chameleon 20 are registered trademarks of Tekelec.

Other product names used herein are for identification purposes only, and may be trademarks of their respective companies.

The hardware, software and documentation comprising the product(s) are provided under a Tekelec limited 12 month warranty. Other than the limited warranties that are expressly stated therein, and without limiting the generality thereof, Tekelec makes no warranty, express or implied, to you or to any other person or entity, concerning the hardware, the software and this documentation. Tekelec will not be liable for incidental, consequential, lost profits, or other similar damages, or for damages resulting from loss of use, data, revenues or time. In no event will Tekelec's liability, regardless of the form of claim, for any damages ever exceed the price/license fee paid for the specific product. You may have other rights which vary from state to state.
# TABLE OF CONTENTS

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LF-1</td>
</tr>
</tbody>
</table>

## CHAPTER 1: INTRODUCTION TO SIMULATION

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Chameleon Simulation Concepts</td>
<td>1.1-1</td>
</tr>
<tr>
<td>Why use Simulation?</td>
<td>1.1-1</td>
</tr>
<tr>
<td>Why BASIC?</td>
<td>1.1-2</td>
</tr>
<tr>
<td>1.2. Simulation Languages</td>
<td>1.2-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>1.2-1</td>
</tr>
<tr>
<td>Simulator Applications</td>
<td>1.2-3</td>
</tr>
<tr>
<td>File Name Extensions</td>
<td>1.2-4</td>
</tr>
</tbody>
</table>

## CHAPTER 2: SIMULATION CONFIGURATION

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Starting Simulation</td>
<td>2.1-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>2.1-1</td>
</tr>
<tr>
<td>Applications Selection Menu</td>
<td>2.1-5</td>
</tr>
<tr>
<td>Page Manipulation Keys</td>
<td>2.1-6</td>
</tr>
<tr>
<td>Simulation Next Steps</td>
<td>2.1-9</td>
</tr>
<tr>
<td>2.2 Parameter Set-Up Menus</td>
<td>2.2-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>2.2-1</td>
</tr>
<tr>
<td>Using Set-Up Menus</td>
<td>2.2-2</td>
</tr>
<tr>
<td>Other Options</td>
<td>2.2-4</td>
</tr>
<tr>
<td>2.3 Save Parameters Menu</td>
<td>2.3-1</td>
</tr>
</tbody>
</table>

## CHAPTER 3: CHAMELEON BASIC

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Chameleon Basic Introduction</td>
<td>3.1-1</td>
</tr>
<tr>
<td>3.2 BASIC Features</td>
<td>3.2-1</td>
</tr>
<tr>
<td>Simulation Buffers</td>
<td>3.2-2</td>
</tr>
<tr>
<td>Mnemonics</td>
<td>3.2-4</td>
</tr>
<tr>
<td>Chameleon Basic Timers</td>
<td>3.2-5</td>
</tr>
<tr>
<td>Arithmetic and Logical Operators</td>
<td>3.2-6</td>
</tr>
<tr>
<td>Variables</td>
<td>3.2-8</td>
</tr>
<tr>
<td>Arrays</td>
<td>3.2-11</td>
</tr>
<tr>
<td>3.3 Introduction to BASIC Programming</td>
<td>3.3-1</td>
</tr>
<tr>
<td>Command Types</td>
<td>3.3-1</td>
</tr>
<tr>
<td>General Syntax Rules</td>
<td>3.3-2</td>
</tr>
<tr>
<td>Command Abbreviations</td>
<td>3.3-4</td>
</tr>
<tr>
<td>Command Errors</td>
<td>3.3-5</td>
</tr>
<tr>
<td>Brief Hands-On Tutorial</td>
<td>3.3-6</td>
</tr>
<tr>
<td>3.4 BASIC Command Index (Functional Listing)</td>
<td>3.4-1</td>
</tr>
<tr>
<td>Miscellaneous Commands</td>
<td>3.4-2</td>
</tr>
<tr>
<td>Data File Commands</td>
<td>3.4-2</td>
</tr>
<tr>
<td>Display and Print Commands</td>
<td>3.4-3</td>
</tr>
<tr>
<td>Function Key Commands</td>
<td>3.4-4</td>
</tr>
<tr>
<td>Loop, Subroutine, and Conditional Commands</td>
<td>3.4-4</td>
</tr>
<tr>
<td>Mnemonic Commands</td>
<td>3.4-5</td>
</tr>
<tr>
<td>Read-Only Variables</td>
<td>3.4-5</td>
</tr>
<tr>
<td>Program Development Commands</td>
<td>3.4-6</td>
</tr>
<tr>
<td>String Commands</td>
<td>3.4-7</td>
</tr>
<tr>
<td>Video Display Commands</td>
<td>3.4-8</td>
</tr>
<tr>
<td>Trace Buffer Commands</td>
<td>3.4-9</td>
</tr>
<tr>
<td>Basic Control and Edit Key Commands</td>
<td>3.4-9</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>@ (ARRAY)</td>
<td>3.5-2</td>
</tr>
<tr>
<td>ABS</td>
<td>3.5-3</td>
</tr>
<tr>
<td>ASC$</td>
<td>3.5-4</td>
</tr>
<tr>
<td>ATIME$</td>
<td>3.5-5</td>
</tr>
<tr>
<td>AUTO</td>
<td>3.5-6</td>
</tr>
<tr>
<td>BCD$</td>
<td>3.5-7</td>
</tr>
<tr>
<td>BLK</td>
<td>3.5-8</td>
</tr>
<tr>
<td>BLKHLF</td>
<td>3.5-9</td>
</tr>
<tr>
<td>BLKREV</td>
<td>3.5-10</td>
</tr>
<tr>
<td>BLKUND</td>
<td>3.5-11</td>
</tr>
<tr>
<td>CALL</td>
<td>3.5-12</td>
</tr>
<tr>
<td>CHAIN</td>
<td>3.5-15</td>
</tr>
<tr>
<td>CHR$</td>
<td>3.5-16</td>
</tr>
<tr>
<td>CLEAR</td>
<td>3.5-17</td>
</tr>
<tr>
<td>CLOSE</td>
<td>3.5-18</td>
</tr>
<tr>
<td>CLS</td>
<td>3.5-19</td>
</tr>
<tr>
<td>COUPLER</td>
<td>3.5-20</td>
</tr>
<tr>
<td>DEC$</td>
<td>3.5-21</td>
</tr>
<tr>
<td>DEFINE</td>
<td>3.5-22</td>
</tr>
<tr>
<td>DEL</td>
<td>3.5-23</td>
</tr>
<tr>
<td>DELETE</td>
<td>3.5-24</td>
</tr>
<tr>
<td>DISP$</td>
<td>3.5-25</td>
</tr>
<tr>
<td>EBC$</td>
<td>3.5-26</td>
</tr>
<tr>
<td>EDIT</td>
<td>3.5-27</td>
</tr>
<tr>
<td>EOF</td>
<td>3.5-29</td>
</tr>
<tr>
<td>ERAEOL</td>
<td>3.5-30</td>
</tr>
<tr>
<td>ERAEOS</td>
<td>3.5-31</td>
</tr>
<tr>
<td>ERASE</td>
<td>3.5-32</td>
</tr>
<tr>
<td>EXIT</td>
<td>3.5-33</td>
</tr>
<tr>
<td>FDEFINE</td>
<td>3.5-34</td>
</tr>
<tr>
<td>FILES</td>
<td>3.5-35</td>
</tr>
<tr>
<td>FLIST</td>
<td>3.5-36</td>
</tr>
<tr>
<td>FLOAD</td>
<td>3.5-37</td>
</tr>
<tr>
<td>FLUSH</td>
<td>3.5-38</td>
</tr>
<tr>
<td>FOR</td>
<td>3.5-39</td>
</tr>
<tr>
<td>FREE</td>
<td>3.5-40</td>
</tr>
<tr>
<td>FSAVE</td>
<td>3.5-41</td>
</tr>
<tr>
<td>GOSUB</td>
<td>3.5-42</td>
</tr>
<tr>
<td>GOTO</td>
<td>3.5-43</td>
</tr>
<tr>
<td>HEX$</td>
<td>3.5-44</td>
</tr>
<tr>
<td>HEX&gt;</td>
<td>3.5-45</td>
</tr>
<tr>
<td>HLF</td>
<td>3.5-46</td>
</tr>
<tr>
<td>HLFUND</td>
<td>3.5-47</td>
</tr>
<tr>
<td>IF</td>
<td>3.5-48</td>
</tr>
<tr>
<td>INKEY$</td>
<td>3.5-50</td>
</tr>
<tr>
<td>INPUT</td>
<td>3.5-51</td>
</tr>
<tr>
<td>$INPUT</td>
<td>3.5-53</td>
</tr>
<tr>
<td>INS</td>
<td>3.5-54</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

BASIC Commands (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR</td>
<td>3.5-55</td>
</tr>
<tr>
<td>KILL</td>
<td>3.5-56</td>
</tr>
<tr>
<td>LEFT$</td>
<td>3.5-57</td>
</tr>
<tr>
<td>LEN</td>
<td>3.5-58</td>
</tr>
<tr>
<td>LET</td>
<td>3.5-59</td>
</tr>
<tr>
<td>LFILES</td>
<td>3.5-60</td>
</tr>
<tr>
<td>LFLIST</td>
<td>3.5-61</td>
</tr>
<tr>
<td>LIST</td>
<td>3.5-62</td>
</tr>
<tr>
<td>LLIST</td>
<td>3.5-63</td>
</tr>
<tr>
<td>LMLIST</td>
<td>3.5-64</td>
</tr>
<tr>
<td>LOAD</td>
<td>3.5-65</td>
</tr>
<tr>
<td>LPRINT</td>
<td>3.5-66</td>
</tr>
<tr>
<td>MENU</td>
<td>3.5-67</td>
</tr>
<tr>
<td>MERGE</td>
<td>3.5-68</td>
</tr>
<tr>
<td>MID$</td>
<td>3.5-69</td>
</tr>
<tr>
<td>MLIST</td>
<td>3.5-70</td>
</tr>
<tr>
<td>MLOAD</td>
<td>3.5-71</td>
</tr>
<tr>
<td>MSAVE</td>
<td>3.5-72</td>
</tr>
<tr>
<td>NEW</td>
<td>3.5-73</td>
</tr>
<tr>
<td>NEXT</td>
<td>3.5-74</td>
</tr>
<tr>
<td>NRM</td>
<td>3.5-75</td>
</tr>
<tr>
<td>OPEN</td>
<td>3.5-76</td>
</tr>
<tr>
<td>PRINT</td>
<td>3.5-78</td>
</tr>
<tr>
<td>READ</td>
<td>3.5-80</td>
</tr>
<tr>
<td>REC</td>
<td>3.5-81</td>
</tr>
<tr>
<td>REM</td>
<td>3.5-82</td>
</tr>
<tr>
<td>RESEQ</td>
<td>3.5-83</td>
</tr>
<tr>
<td>RETURN</td>
<td>3.5-85</td>
</tr>
<tr>
<td>REV</td>
<td>3.5-86</td>
</tr>
<tr>
<td>REVHFLF</td>
<td>3.5-87</td>
</tr>
<tr>
<td>REVUND</td>
<td>3.5-88</td>
</tr>
<tr>
<td>RIGHTS$</td>
<td>3.5-89</td>
</tr>
<tr>
<td>RND</td>
<td>3.5-90</td>
</tr>
<tr>
<td>RUN</td>
<td>3.5-91</td>
</tr>
<tr>
<td>SAVE</td>
<td>3.5-92</td>
</tr>
<tr>
<td>SET</td>
<td>3.5-93</td>
</tr>
<tr>
<td>SETUP</td>
<td>3.5-94</td>
</tr>
<tr>
<td>SIZE</td>
<td>3.5-95</td>
</tr>
<tr>
<td>STOP</td>
<td>3.5-96</td>
</tr>
<tr>
<td>TEST</td>
<td>3.5-97</td>
</tr>
<tr>
<td>TFREE</td>
<td>3.5-98</td>
</tr>
<tr>
<td>TIME</td>
<td>3.5-99</td>
</tr>
<tr>
<td>TIMES$</td>
<td>3.5-100</td>
</tr>
<tr>
<td>TLOAD</td>
<td>3.5-101</td>
</tr>
<tr>
<td>TPRINT</td>
<td>3.5-102</td>
</tr>
<tr>
<td>TROFF</td>
<td>3.5-103</td>
</tr>
<tr>
<td>TRON</td>
<td>3.5-104</td>
</tr>
<tr>
<td>TSAVE</td>
<td>3.5-105</td>
</tr>
<tr>
<td>UND</td>
<td>3.5-106</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS

#### BASIC Commands (continued)
- VAL .......................................................... 3.5–107
- WRITE ......................................................... 3.5–108
- XYPILOT .................................................. 3.5–109

#### CHAPTER 4 FRAMEM

4.1 Introduction to Framem ........................................ 4.1–1
4.2 FRAMEM Command Index (Functional Listing) .......... 4.2–1
  Mnemonic Commands ........................................ 4.2–2
  Addressing Commands ....................................... 4.2–2
  Miscellaneous Commands .................................... 4.2–2
  Transmission and Reception Variables ....................... 4.2–3
4.3 FRAMEM Commands (Alphabetical Listing) ................. 4.3–1
  ABORTRAN .................................................. 4.3–2
  BADTRAN ................................................... 4.3–3
  CRC ............................................................. 4.3–4
  DEFINE ....................................................... 4.3–5
  DEFSUB ....................................................... 4.3–6
  EXTEND ...................................................... 4.3–7
  GET ............................................................. 4.3–8
  MOD ............................................................. 4.3–9
  NORM .......................................................... 4.3–10
  PUT ............................................................. 4.3–11
  REC ............................................................. 4.3–12
  RXADDR ..................................................... 4.3–13
  RXC/R ........................................................ 4.3–14
  RXDIAG ...................................................... 4.3–15
  RXFCCTL ................................................... 4.3–16
  RXFRLEN .................................................... 4.3–17
  RXN(R) ........................................................ 4.3–18
  RXN(S) ........................................................ 4.3–19
  RXV(R) ........................................................ 4.3–20
  RXV(S) ........................................................ 4.3–21
  RXFRLEN .................................................... 4.3–24
  STATUS ...................................................... 4.3–25
  TPRINT ...................................................... 4.3–26
  TRAN .......................................................... 4.3–27
  TXADDR ..................................................... 4.3–28
  TXC/R ........................................................ 4.3–29
  TXDIAG ...................................................... 4.3–30
  TXFCCTL ................................................... 4.3–31
  TXFIELD ..................................................... 4.3–32
  TXN(R) ........................................................ 4.3–33
  TXN(S) ........................................................ 4.3–34
  TXV(R) ........................................................ 4.3–35
  TXV(S) ........................................................ 4.3–36

TOC–4 Version 4.9
# TABLE OF CONTENTS

4.4 FRAMEM HDLC/SDL ...................................................... 4.4-1  
- Parameter Set-Up Menu ........................................... 4.4-2  
- Transparent Bisync Set-Up Menu ................................. 4.4-4  
- Mnemonic Table .................................................... 4.4-5  
- Commands ......................................................... 4.4-6  
  - TPRINT ......................................................... 4.4-7  
4.4 Sample Programs .................................................... 4.4-8

4.5 FRAMEM LAPD ............................................................ 4.5-1  
- Introduction ....................................................... 4.5-1  
- LAPD Frame .......................................................... 4.5-1  
- Capabilities of FRAMEM LAPD ..................................... 4.5-2  
- Parameter Set-Up Menu ............................................ 4.5-3  
- Mnemonic Table ..................................................... 4.5-5  
- Command Index (Functional Listing) .............................. 4.5-6  
  - Transmission and Reception Commands ........................ 4.5-6  
  - Miscellaneous Commands ....................................... 4.5-6  
- Commands (Alphabetical Listing) ................................. 4.5-7  
  - DEFINE .......................................................... 4.5-7  
  - FILL .............................................................. 4.5-8  
  - RXCR ............................................................ 4.5-9  
  - RXSAPI ....................................................... 4.5-10  
  - RXTE ............................................................ 4.5-11  
  - TPRINT .......................................................... 4.5-12  
  - TXCR ............................................................. 4.5-13  
  - TXSAPI ....................................................... 4.5-14  
  - TXTEI ........................................................... 4.5-15  
- Sample Programs ..................................................... 4.5-16

4.6 FRAMEM DMI .............................................................. 4.6-1  
- Introduction ....................................................... 4.6-1  
- Setting Up FRAMEM DMI ............................................ 4.6-1  
- Command Index (Functional Listing) .............................. 4.6-3  
  - Read-Only Variables .......................................... 4.6-3  
  - Commands ....................................................... 4.6-4  
- FRAMEM DMI Commands (Alphabetical Listing) ................... 4.6-5  
  - CAUSE .......................................................... 4.6-5  
  - CHADMIN ...................................................... 4.6-6  
  - CONNECT ...................................................... 4.6-7  
  - DCALLED ....................................................... 4.6-8  
  - DCALLING .................................................... 4.6-9  
  - DISCONNECT ................................................... 4.6-10  
  - DMATCH .......................................................... 4.6-11  
  - DTIMERS ....................................................... 4.6-12  
  - GLARE .......................................................... 4.6-15  
  - MATCH .......................................................... 4.6-16  
  - OUTNUM .......................................................... 4.6-17  
  - RESET ........................................................... 4.6-18  
  - RESPTIME ...................................................... 4.6-19  
  - STATE ........................................................... 4.6-20  
  - STATUS ........................................................... 4.6-21  
  - TPRINT .......................................................... 4.6-22
# TABLE OF CONTENTS

Sample Programs ........................................................ 4.6-23

## CHAPTER 5: SIMP/L

5.1 Introduction to SIMP/L ............................................. 5.1-1
5.2 SIMP/L Command Index (Functional Listing) .............. 5.2-1
  Miscellaneous Commands ........................................... 5.2-1
  Print and Display Commands ....................................... 5.2-1
  Transmission and Reception Commands ....................... 5.2-2
  Read–Only Variables ............................................... 5.2-2

5.3 SIMP/L Commands (Alphabetical Listing)
  BREAK ................................................................. 5.3-2
  BUFFER ..................................................................... 5.3-3
  BUILD ...................................................................... 5.3-4
  DEFINE .................................................................... 5.3-5
  LENGTH ..................................................................... 5.3-6
  LNKSTAT ................................................................. 5.3-7
  LRDISPF ................................................................. 5.3-8
  LTDISPF ................................................................. 5.3-9
  RDISPF ..................................................................... 5.3-10
  REC ......................................................................... 5.3-11
  SET .......................................................................... 5.3-11
  SLOF ......................................................................... 5.3-13
  SLON ......................................................................... 5.3-14
  STATUS ..................................................................... 5.3-15
  TDISPF ...................................................................... 5.3-16
  TPRINT ....................................................................... 5.3-17
  TRAN ......................................................................... 5.3-18

5.4 SIMP/L HDLC ............................................................... 5.4-1
  General Characteristics .................................................. 5.4-1
  HDLC Frame Formats ................................................... 5.4-1
  Parameter Set-Up Menu .................................................. 5.4-3
  Mnemonic Table ............................................................ 5.4-6
  SIMP/L HDLC Command Index (Functional Listing) ...... 5.4-7
    Miscellaneous Commands ........................................... 5.4-7
    Read–Only Variables ............................................... 5.4-7
  Commands (Alphabetical Listing)
    LNKSTAT ................................................................. 5.4-8
    SET .......................................................................... 5.4-10
    STATUS ..................................................................... 5.4-12
    TPRINT ....................................................................... 5.4-13

5.5 SIMP/L SDLC ............................................................... 5.5-1
  General Characteristics .................................................. 5.5-1
  SDLC Frame Format ...................................................... 5.5-2
  SDLC 8–Bit Control Field Frames ................................... 5.5-3
  Parameter Set-Up Menu .................................................. 5.5-4
  Mnemonics ................................................................. 5.5-7
  SIMP/L SDLC Command Index (Functional Listing)
    Read–Only Variables ............................................... 5.5-8
    Transmission and Reception Commands ............... 5.5-8
    Miscellaneous Commands ........................................... 5.5-8

TOC-6  Version 4.9
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands (Alphabetical Listing)</td>
<td>5.5-9</td>
</tr>
<tr>
<td>LNKSTAT</td>
<td>5.5-9</td>
</tr>
<tr>
<td>NSI</td>
<td>5.5-10</td>
</tr>
<tr>
<td>SET T1</td>
<td>5.5-11</td>
</tr>
<tr>
<td>SET N2</td>
<td>5.5-11</td>
</tr>
<tr>
<td>SET T1</td>
<td>5.5-11</td>
</tr>
<tr>
<td>SET ADDR</td>
<td>5.5-11</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.5-12</td>
</tr>
<tr>
<td>TEST</td>
<td>5.5-13</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.5-14</td>
</tr>
<tr>
<td>XID</td>
<td>5.5-15</td>
</tr>
<tr>
<td>XIDFLD</td>
<td>5.5-16</td>
</tr>
<tr>
<td>Sample Programs</td>
<td>5.5-17</td>
</tr>
<tr>
<td>5.6 SIMP/L LAPD</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>5.6-1</td>
</tr>
<tr>
<td>Control Features</td>
<td>5.6-2</td>
</tr>
<tr>
<td>Multiple Addressing</td>
<td>5.6-3</td>
</tr>
<tr>
<td>Frame Status Byte</td>
<td>5.6-3</td>
</tr>
<tr>
<td>XID Frames</td>
<td>5.6-4</td>
</tr>
<tr>
<td>LAPD Frame Format</td>
<td>5.6-5</td>
</tr>
<tr>
<td>Frame Specifications (Mod 8)</td>
<td>5.6-6</td>
</tr>
<tr>
<td>Frame Specifications (Mod 128)</td>
<td>5.6-7</td>
</tr>
<tr>
<td>Parameter Set-Up Menu</td>
<td>5.6-8</td>
</tr>
<tr>
<td>Command Index (Functional Listing)</td>
<td>5.6-11</td>
</tr>
<tr>
<td>Read-Only Variables</td>
<td>5.6-11</td>
</tr>
<tr>
<td>Parameter Commands</td>
<td>5.6-11</td>
</tr>
<tr>
<td>Transmission Commands</td>
<td>5.6-12</td>
</tr>
<tr>
<td>Commands (Alphabetical Listing)</td>
<td>5.6-13</td>
</tr>
<tr>
<td>EXTEND</td>
<td>5.6-13</td>
</tr>
<tr>
<td>FRSTAT</td>
<td>5.6-14</td>
</tr>
<tr>
<td>LNKSTAT</td>
<td>5.6-15</td>
</tr>
<tr>
<td>MOD</td>
<td>5.6-17</td>
</tr>
<tr>
<td>SET N200</td>
<td>5.6-18</td>
</tr>
<tr>
<td>SET N201</td>
<td>5.6-18</td>
</tr>
<tr>
<td>SET Network</td>
<td>5.6-18</td>
</tr>
<tr>
<td>SET Subscriber</td>
<td>5.6-18</td>
</tr>
<tr>
<td>SET SAPI</td>
<td>5.6-18</td>
</tr>
<tr>
<td>SET TEI</td>
<td>5.6-19</td>
</tr>
<tr>
<td>SET T200</td>
<td>5.6-19</td>
</tr>
<tr>
<td>SET T203</td>
<td>5.6-19</td>
</tr>
<tr>
<td>SET Window</td>
<td>5.6-20</td>
</tr>
<tr>
<td>SET (fnctn)</td>
<td>5.6-21</td>
</tr>
<tr>
<td>SET CONFIG</td>
<td>5.6-22</td>
</tr>
<tr>
<td>SET RSAPI</td>
<td>5.6-24</td>
</tr>
<tr>
<td>SET RTEI</td>
<td>5.6-25</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.6-26</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.6-27</td>
</tr>
<tr>
<td>TRUI</td>
<td>5.6-29</td>
</tr>
<tr>
<td>TRXIDC</td>
<td>5.6-30</td>
</tr>
<tr>
<td>TRXIDR</td>
<td>5.6-31</td>
</tr>
</tbody>
</table>

TOC-7 • Version 4.9
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNEXTEND</td>
<td>5.6-32</td>
</tr>
<tr>
<td>Mnemonics</td>
<td>5.6-33</td>
</tr>
<tr>
<td>Q.931 Message Format</td>
<td>5.6-33</td>
</tr>
<tr>
<td>Mnemonic Table</td>
<td>5.6-35</td>
</tr>
<tr>
<td>Frame Level Configuration</td>
<td>5.6-35</td>
</tr>
<tr>
<td>Configuration Checklist</td>
<td>5.6-40</td>
</tr>
<tr>
<td>Sample Programs</td>
<td>5.6-41</td>
</tr>
<tr>
<td>Converting Programs to Extended SIMP/L LAPD</td>
<td>5.6-49</td>
</tr>
<tr>
<td><strong>5.7 Multi-Link SIMP/L LAP</strong></td>
<td>5.7-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>5.7-1</td>
</tr>
<tr>
<td>Link Selection</td>
<td>5.7-2</td>
</tr>
<tr>
<td>General Notes</td>
<td>5.7-2</td>
</tr>
<tr>
<td>SIMP/L LAPD Commands</td>
<td>5.7-3</td>
</tr>
<tr>
<td>Multi-Link SIMP/L Commands</td>
<td>5.7-4</td>
</tr>
<tr>
<td>FRELNK</td>
<td>5.7-5</td>
</tr>
<tr>
<td>FRSTAT</td>
<td>5.7-6</td>
</tr>
<tr>
<td>RECLNK</td>
<td>5.7-7</td>
</tr>
<tr>
<td>SET LINK</td>
<td>5.7-8</td>
</tr>
<tr>
<td>SET SAPI</td>
<td>5.7-9</td>
</tr>
<tr>
<td>SET TEI</td>
<td>5.7-10</td>
</tr>
<tr>
<td>SET TG</td>
<td>5.7-11</td>
</tr>
<tr>
<td>STATE</td>
<td>5.7-12</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.7-13</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.7-14</td>
</tr>
<tr>
<td>Sample Program</td>
<td>5.7-15</td>
</tr>
<tr>
<td><strong>5.8 SIMP/L V.12</strong></td>
<td>5.8-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>5.8-1</td>
</tr>
<tr>
<td>V.120 Address</td>
<td>5.8-3</td>
</tr>
<tr>
<td>Commands</td>
<td>5.8-1</td>
</tr>
<tr>
<td>FRELNK</td>
<td>5.8-5</td>
</tr>
<tr>
<td>FRSTAT</td>
<td>5.8-6</td>
</tr>
<tr>
<td>RECLNK</td>
<td>5.8-7</td>
</tr>
<tr>
<td>RTRAN</td>
<td>5.8-8</td>
</tr>
<tr>
<td>SET LINK</td>
<td>5.8-9</td>
</tr>
<tr>
<td>SET LLI</td>
<td>5.8-10</td>
</tr>
<tr>
<td>STATE</td>
<td>5.8-11</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.8-12</td>
</tr>
<tr>
<td>Sample Program</td>
<td>5.8-13</td>
</tr>
</tbody>
</table>

**CHAPTER 6: BSC**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Chameleon BSC Fundamentals</td>
<td>6.1-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>6.1-1</td>
</tr>
<tr>
<td>BSC Applications</td>
<td>6.1-1</td>
</tr>
<tr>
<td>BSC Frame</td>
<td>6.1-2</td>
</tr>
<tr>
<td>6.2 Parameter Set-Up Menu</td>
<td>6.2-1</td>
</tr>
<tr>
<td>6.3 Mnemonics</td>
<td>6.3-1</td>
</tr>
<tr>
<td>6.4 Command Index (Functional Listing)</td>
<td>6.4-1</td>
</tr>
<tr>
<td>Transmission and Reception Commands</td>
<td>6.4-1</td>
</tr>
<tr>
<td>Miscellaneous Commands</td>
<td>6.4-1</td>
</tr>
<tr>
<td>6.5 Commands (Alphabetical List)</td>
<td>6.5-1</td>
</tr>
<tr>
<td>CRC16</td>
<td>6.5-2</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>6.5-3</td>
</tr>
<tr>
<td>LRC</td>
<td>6.5-4</td>
</tr>
<tr>
<td>REC</td>
<td>6.5-5</td>
</tr>
<tr>
<td>RXLENGTH</td>
<td>6.5-6</td>
</tr>
<tr>
<td>TPRINT</td>
<td>6.5-7</td>
</tr>
<tr>
<td>TRAN</td>
<td>6.5-8</td>
</tr>
<tr>
<td>TXBUFFER</td>
<td>6.5-9</td>
</tr>
<tr>
<td>TXSTATUS</td>
<td>6.5-10</td>
</tr>
<tr>
<td>6.6 Sample Programs</td>
<td>6.6-1</td>
</tr>
<tr>
<td><strong>CHAPTER 7: ASYNC</strong></td>
<td></td>
</tr>
<tr>
<td>7.1 Introduction to Chameleon Async</td>
<td>7.1-1</td>
</tr>
<tr>
<td>7.2 Parameter Set-Up Menu</td>
<td>7.2-1</td>
</tr>
<tr>
<td>7.3 Mnemonic Table</td>
<td>7.3-2</td>
</tr>
<tr>
<td>7.4 Command Index (Functional Listing)</td>
<td>7.4-1</td>
</tr>
<tr>
<td>Read-Only Variables</td>
<td>7.4-1</td>
</tr>
<tr>
<td>Transmission and Reception Commands</td>
<td>7.4-1</td>
</tr>
<tr>
<td>7.5 Commands (Alphabetical Listing)</td>
<td>7.5-1</td>
</tr>
<tr>
<td>BREAK</td>
<td>7.5-2</td>
</tr>
<tr>
<td>CRC16</td>
<td>7.5-3</td>
</tr>
<tr>
<td>FOXMESS</td>
<td>7.5-4</td>
</tr>
<tr>
<td>FRAMING</td>
<td>7.5-5</td>
</tr>
<tr>
<td>LENGTH</td>
<td>7.5-6</td>
</tr>
<tr>
<td>LRC</td>
<td>7.5-7</td>
</tr>
<tr>
<td>PARITY</td>
<td>7.5-8</td>
</tr>
<tr>
<td>REC</td>
<td>7.5-9</td>
</tr>
<tr>
<td>RXBREAK</td>
<td>7.5-10</td>
</tr>
<tr>
<td>TPRINT</td>
<td>7.5-11</td>
</tr>
<tr>
<td>TRAN</td>
<td>7.5-12</td>
</tr>
<tr>
<td>7.6 Sample Programs</td>
<td>7.6-1</td>
</tr>
<tr>
<td><strong>CHAPTER 8: SITREX</strong></td>
<td></td>
</tr>
<tr>
<td>8.1 Introduction to Sitrex</td>
<td>8.1-1</td>
</tr>
<tr>
<td>Sitrex Standards</td>
<td>8.1-1</td>
</tr>
<tr>
<td>Sitrex Environment</td>
<td>8.1-2</td>
</tr>
<tr>
<td>Other Sitrex Features</td>
<td>8.1-10</td>
</tr>
<tr>
<td>8.2 Automatic X.25 Simulator</td>
<td>8.2-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>8.2-1</td>
</tr>
<tr>
<td>Accessing SITREX</td>
<td>8.2-1</td>
</tr>
<tr>
<td>Automatic X.25 Simulator Commands</td>
<td>8.2-2</td>
</tr>
<tr>
<td>Pseudo-Users</td>
<td>8.2-3</td>
</tr>
<tr>
<td>Controlling X.25 Levels</td>
<td>8.2-4</td>
</tr>
<tr>
<td>8.3 Sitrex Menus</td>
<td>8.3-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>8.3-1</td>
</tr>
<tr>
<td>Physical Level Menu</td>
<td>8.3-2</td>
</tr>
<tr>
<td>Frame &amp; Packet Level Menu</td>
<td>8.3-4</td>
</tr>
<tr>
<td>Facilities Menu</td>
<td>8.3-6</td>
</tr>
<tr>
<td>8.4 SITREX Trace Buffer</td>
<td>8.4-1</td>
</tr>
<tr>
<td>Activating the Trace</td>
<td>8.4-1</td>
</tr>
<tr>
<td>Trace Commands</td>
<td>8.4-2</td>
</tr>
</tbody>
</table>

TOC-9• Version 4.9
## TABLE OF CONTENTS

Trace Interpretation ........................................................ 8.4-4

### 8.5 SITREX Command Mode

- Introduction .............................................................. 8.5-1
- Command Groups .......................................................... 8.5-2
- General Syntax Rules .................................................. 8.5-3
- Syntax Error .................................................................. 8.5-6

### 8.6 SITREX Command Index

- Alphabetical Command Index ........................................... 8.6-2
- Functional Command Index .............................................. 8.6-7

### 8.7 Commands .............................................................. 8.7-1

#### Frame Level Commands
- Send User-Defined Frame ............................................. 8.7-2
- Send Unnumbered Commands .......................................... 8.7-3
- Send Unnumbered Responses ......................................... 8.7-4
- Send Numbered Commands ............................................ 8.7-5
- Send Numbered Responses ............................................ 8.7-6
- Send Information Frame ................................................ 8.7-7

#### Packet Level Commands
- Send User-Defined Packet ............................................ 8.7-9
- Send Call/Call Confirmation Packet ................................. 8.7-10
- Send Supervisory Packet ............................................... 8.7-11
- Send Restart/Restart Confirmation Packet ......................... 8.7-12
- Send Clear/Reset/Interrupt and Confirmation Packets ........... 8.7-13
- Send Data Packets ...................................................... 8.7-14
- Send Diagnostic Packet ................................................ 8.7-15

#### Parameter Commands
- Set Frame Level ......................................................... 8.7-16
- Set Packet Level ......................................................... 8.7-17
- Set Link Level ............................................................ 8.7-19
- Force Link On ................................................................ 8.7-20
- Transmit CRC .............................................................. 8.7-21
- Set Primary/Secondary Address ....................................... 8.7-22
- Set Frame and Packet Window Size .................................. 8.7-23
- Set Data Packet Length ................................................. 8.7-24
- Set Frame and Packet State Variables .............................. 8.7-25
- Let Logical Channel Group Number ................................ 8.7-26
- Set Interface Leads ...................................................... 8.7-27
- Test Interface Leads ..................................................... 8.7-28
- Set Timers ................................................................. 8.7-29
- Set Counters .............................................................. 8.7-30
- Set Pseudo–User Type .................................................. 8.7-31
- Set Up PVCs and SVCs .................................................. 8.7-32

#### Wait Commands
- Wait for Frame or Packet ............................................... 8.7-33
- Wait for Byte Mask Configuration ..................................... 8.7-34
- Set Wait Jump Addresses ............................................... 8.7-35
- Set Watchdog Timer ..................................................... 8.7-36

#### Loop, Jump, Reinitialization Commands
- Set up Program Loop ..................................................... 8.7-37
- Conditional Jump (IF) .................................................... 8.7-38

TOC-10• Version 4.9
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconditional Jump (GOTO, GOSUB)</td>
<td>8.7-46</td>
</tr>
<tr>
<td>Reinitialization</td>
<td>8.7-47</td>
</tr>
<tr>
<td>Program Management Commands</td>
<td>8.7-48</td>
</tr>
<tr>
<td>Chain Programs</td>
<td>8.7-49</td>
</tr>
<tr>
<td>Load Program</td>
<td>8.7-50</td>
</tr>
<tr>
<td>Remark</td>
<td>8.7-51</td>
</tr>
<tr>
<td>New Program</td>
<td>8.7-52</td>
</tr>
<tr>
<td>Run Program</td>
<td>8.7-53</td>
</tr>
<tr>
<td>Save Program</td>
<td>8.7-54</td>
</tr>
<tr>
<td>Numeric Variable Commands</td>
<td>8.7-55</td>
</tr>
<tr>
<td>Variable Operations</td>
<td>8.7-56</td>
</tr>
<tr>
<td>Shift and Rotate Variable Contents</td>
<td>8.7-57</td>
</tr>
<tr>
<td>Keyboard Input</td>
<td>8.7-58</td>
</tr>
<tr>
<td>Test Variables</td>
<td>8.7-59</td>
</tr>
<tr>
<td>Message Buffer Commands</td>
<td>8.7-60</td>
</tr>
<tr>
<td>Define Message for Message Buffer</td>
<td>8.7-60</td>
</tr>
<tr>
<td>Message Buffer Byte Extraction</td>
<td>8.7-62</td>
</tr>
<tr>
<td>Assign Message Buffer Length</td>
<td>8.7-63</td>
</tr>
<tr>
<td>Test Message Buffer Contents</td>
<td>8.7-64</td>
</tr>
<tr>
<td>Transmit Message</td>
<td>8.7-65</td>
</tr>
<tr>
<td>Trace Buffer Commands</td>
<td>8.7-66</td>
</tr>
<tr>
<td>Display Trace Buffer</td>
<td>8.7-67</td>
</tr>
<tr>
<td>Load Trace File From Disk</td>
<td>8.7-68</td>
</tr>
<tr>
<td>Print Trace Buffer</td>
<td>8.7-69</td>
</tr>
<tr>
<td>Save Trace to Disk</td>
<td>8.7-70</td>
</tr>
<tr>
<td>Set Trace Buffer On/Off</td>
<td>8.7-71</td>
</tr>
<tr>
<td>Clear Trace Buffer</td>
<td>8.7-72</td>
</tr>
<tr>
<td>Set Trace Length</td>
<td>8.7-73</td>
</tr>
<tr>
<td>Display and Print Commands</td>
<td>8.7-74</td>
</tr>
<tr>
<td>Display User Parameters</td>
<td>8.7-75</td>
</tr>
<tr>
<td>Display Screen Messages and Prompts</td>
<td>8.7-77</td>
</tr>
<tr>
<td>List Program File</td>
<td>8.7-78</td>
</tr>
<tr>
<td>Print Program File</td>
<td>8.7-79</td>
</tr>
<tr>
<td>Output Message to Printer or Remote Device</td>
<td>8.7-80</td>
</tr>
<tr>
<td>8.8 SITREX Error Codes</td>
<td>8.8-1</td>
</tr>
<tr>
<td>8.9 Sample Program</td>
<td>8.9-1</td>
</tr>
<tr>
<td>8.10 Simulator Reactions</td>
<td>8.10-1</td>
</tr>
<tr>
<td>8.11 Flow Charts</td>
<td>8.11-1</td>
</tr>
</tbody>
</table>

INDEX
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2-1</td>
<td>The Chameleon BASIC Family</td>
<td>1.2-2</td>
</tr>
<tr>
<td>1.2-2</td>
<td>Simulation Languages</td>
<td>1.2-2</td>
</tr>
<tr>
<td>1.2-3</td>
<td>Simulation Directories and File Extensions</td>
<td>1.2-4</td>
</tr>
<tr>
<td>2.1-1</td>
<td>Menu Setup Mode</td>
<td>2.1-1</td>
</tr>
<tr>
<td>2.1-2</td>
<td>Chameleon Protocol Setup Menu</td>
<td>2.1-2</td>
</tr>
<tr>
<td>2.1-3</td>
<td>Applications Selection Menu</td>
<td>2.1-5</td>
</tr>
<tr>
<td>2.1-4</td>
<td>Page Manipulation Keys</td>
<td>2.1-7</td>
</tr>
<tr>
<td>2.1-5</td>
<td>FRAMEM Parameter Set-Up Menu</td>
<td>2.1-8</td>
</tr>
<tr>
<td>2.1-6</td>
<td>Simulation Next Steps</td>
<td>2.1-10</td>
</tr>
<tr>
<td>2.2-1</td>
<td>FRAMEM Parameter Set-Up Menu</td>
<td>2.2-2</td>
</tr>
<tr>
<td>2.2-2</td>
<td>FRAMEM HDLC Simulation Parameters</td>
<td>2.2-3</td>
</tr>
<tr>
<td>2.3-1</td>
<td>Save Parameters Menu</td>
<td>2.3-1</td>
</tr>
<tr>
<td>3.4-1</td>
<td>Miscellaneous Commands</td>
<td>3.4-2</td>
</tr>
<tr>
<td>3.4-2</td>
<td>Data File Commands</td>
<td>3.4-2</td>
</tr>
<tr>
<td>3.4-3</td>
<td>Display and Print Commands</td>
<td>3.4-3</td>
</tr>
<tr>
<td>3.4-4</td>
<td>Function Key Commands</td>
<td>3.4-4</td>
</tr>
<tr>
<td>3.4-5</td>
<td>Loop, Subroutine, and Conditional Commands</td>
<td>3.4-4</td>
</tr>
<tr>
<td>3.4-6</td>
<td>Mnemonic Commands</td>
<td>3.4-5</td>
</tr>
<tr>
<td>3.4-7</td>
<td>Read Only Variables</td>
<td>3.4-5</td>
</tr>
<tr>
<td>3.4-8</td>
<td>Program Development Commands</td>
<td>3.4-6</td>
</tr>
<tr>
<td>3.4-9</td>
<td>String Commands</td>
<td>3.4-7</td>
</tr>
<tr>
<td>3.4-10</td>
<td>Video Display Commands</td>
<td>3.4-8</td>
</tr>
<tr>
<td>3.4-11</td>
<td>Trace Buffer Commands</td>
<td>3.4-9</td>
</tr>
<tr>
<td>3.5-1</td>
<td>SET Physical Interface Options</td>
<td>3.5-93</td>
</tr>
<tr>
<td>3.5-2</td>
<td>TEST Physical Interface Options</td>
<td>3.5-97</td>
</tr>
<tr>
<td>4.1-1</td>
<td>Standard ADCCP Frame</td>
<td>4.1-2</td>
</tr>
<tr>
<td>4.2-1</td>
<td>FRAMEM Transmission and Reception Commands</td>
<td>4.2-1</td>
</tr>
<tr>
<td>4.2-2</td>
<td>FRAMEM Mnemonic Commands</td>
<td>4.2-2</td>
</tr>
<tr>
<td>4.2-3</td>
<td>FRAMEM Addressing Commands</td>
<td>4.2-2</td>
</tr>
<tr>
<td>4.2-4</td>
<td>FRAMEM Miscellaneous Commands</td>
<td>4.2-2</td>
</tr>
<tr>
<td>4.2-5</td>
<td>FRAMEM Transmission and Reception Variables</td>
<td>4.2-3</td>
</tr>
<tr>
<td>4.4-1</td>
<td>FRAMEM HDLC-SDLC Parameter Set-Up Menu</td>
<td>4.4-2</td>
</tr>
<tr>
<td>4.4-2</td>
<td>Xparent Bisync Parameters</td>
<td>4.4-4</td>
</tr>
<tr>
<td>4.4-3</td>
<td>FRAMEM HDLC-SDLC Mnemonic Table</td>
<td>4.4-5</td>
</tr>
<tr>
<td>4.5-1</td>
<td>LAPD Frame Format</td>
<td>4.5-1</td>
</tr>
<tr>
<td>4.5-2</td>
<td>FRAMEM LAPD Parameter Set-Up Menu</td>
<td>4.5-3</td>
</tr>
<tr>
<td>4.5-3</td>
<td>FRAMEM LAPD Mnemonic Table</td>
<td>4.5-5</td>
</tr>
<tr>
<td>4.5-4</td>
<td>FRAMEM LAPD Transmission and Reception Variables</td>
<td>4.5-6</td>
</tr>
<tr>
<td>4.5-5</td>
<td>FRAMEM LAPD Miscellaneous Commands</td>
<td>4.5-6</td>
</tr>
<tr>
<td>4.6-1</td>
<td>DMI Screen</td>
<td>4.6-1</td>
</tr>
<tr>
<td>4.6-2</td>
<td>FRAMEM DMI Read-Only Variables</td>
<td>4.6-3</td>
</tr>
<tr>
<td>4.6-3</td>
<td>FRAMEM DMI Commands</td>
<td>4.6-4</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6-4</td>
<td>FRAMEM DMI Setup Mode Bit Values</td>
<td>4.6-6</td>
</tr>
<tr>
<td>4.6-5</td>
<td>FRAMEM DMI Timers</td>
<td>4.6-12</td>
</tr>
<tr>
<td>4.6-6</td>
<td>Wink (Normal) Time Period</td>
<td>4.6-13</td>
</tr>
<tr>
<td>4.6-7</td>
<td>Wink (Abnormal) Time Period</td>
<td>4.6-14</td>
</tr>
<tr>
<td>4.6-8</td>
<td>Auto (Normal) Time Period</td>
<td>4.6-14</td>
</tr>
<tr>
<td>5.2-1</td>
<td>SIMP/L Miscellaneous Commands</td>
<td>5.2-1</td>
</tr>
<tr>
<td>5.2-2</td>
<td>SIMP/L Print and Display Commands</td>
<td>5.2-1</td>
</tr>
<tr>
<td>5.2-3</td>
<td>SIMP/L Transmission and Reception Commands</td>
<td>5.2-2</td>
</tr>
<tr>
<td>5.2-4</td>
<td>SIMP/L Read-Only Variables</td>
<td>5.2-2</td>
</tr>
<tr>
<td>5.4-1</td>
<td>Frame Format</td>
<td>5.4-1</td>
</tr>
<tr>
<td>5.4-2</td>
<td>HDLC Frame Format</td>
<td>5.4-2</td>
</tr>
<tr>
<td>5.4-3</td>
<td>SIMP/L HDLC Parameter Set-Up Menu</td>
<td>5.4-3</td>
</tr>
<tr>
<td>5.4-4</td>
<td>SIMP/L HDLC Mnemonics</td>
<td>5.4-6</td>
</tr>
<tr>
<td>5.4-5</td>
<td>SIMP/L HDLC Miscellaneous Commands</td>
<td>5.4-7</td>
</tr>
<tr>
<td>5.4-6</td>
<td>SIMP/L HDLC Read-Only Variables</td>
<td>5.4-7</td>
</tr>
<tr>
<td>5.4-7</td>
<td>SIMP/L HDLC Link Status Values</td>
<td>5.4-8</td>
</tr>
<tr>
<td>5.5-1</td>
<td>SDLC Frame Format</td>
<td>5.5-2</td>
</tr>
<tr>
<td>5.5-2</td>
<td>HDLC Frame Formats (8-bit Control Field)</td>
<td>5.5-3</td>
</tr>
<tr>
<td>5.5-3</td>
<td>SIMP/L SDLC Parameter Set-Up Menu</td>
<td>5.5-4</td>
</tr>
<tr>
<td>5.5-4</td>
<td>SIMP/L SDLC Read-Only Variables</td>
<td>5.5-8</td>
</tr>
<tr>
<td>5.5-5</td>
<td>SIMP/L SDLC Transmission and Reception Commands</td>
<td>5.5-8</td>
</tr>
<tr>
<td>5.5-6</td>
<td>SIMP/L SDLC Miscellaneous Commands</td>
<td>5.5-8</td>
</tr>
<tr>
<td>5.5-7</td>
<td>SIMP/L SDLC Link Status Values (Primary)</td>
<td>5.5-9</td>
</tr>
<tr>
<td>5.5-8</td>
<td>SIMP/L SDLC Link Status Values (Secondary)</td>
<td>5.5-9</td>
</tr>
<tr>
<td>5.5-9</td>
<td>SNA FID Type 2 Transmission Header</td>
<td>5.5-17</td>
</tr>
<tr>
<td>5.5-10</td>
<td>Sample Program Mnemonics</td>
<td>5.5-24</td>
</tr>
<tr>
<td>5.6-1</td>
<td>Frame Format</td>
<td>5.6-5</td>
</tr>
<tr>
<td>5.6-2</td>
<td>SIMP/L LAPD Frame Specifications (Mod 8)</td>
<td>5.6-6</td>
</tr>
<tr>
<td>5.6-3</td>
<td>SIMP/L LAPD Frame Specifications (Mod 128)</td>
<td>5.6-7</td>
</tr>
<tr>
<td>5.6-4</td>
<td>SIMP/L LAPD Parameter Set-Up Menu</td>
<td>5.6-8</td>
</tr>
<tr>
<td>5.6-5</td>
<td>SIMP/L LAPD Read-Only Variables</td>
<td>5.6-11</td>
</tr>
<tr>
<td>5.6-6</td>
<td>SIMP/L LAPD Parameter Commands</td>
<td>5.6-11</td>
</tr>
<tr>
<td>5.6-7</td>
<td>SIMP/L LAPD Transmission Commands</td>
<td>5.6-12</td>
</tr>
<tr>
<td>5.6-8</td>
<td>SIMP/L LAPD Link Status Values</td>
<td>5.6-15</td>
</tr>
<tr>
<td>5.6-9</td>
<td>SIMP/L LAPD User Control Mnemonics</td>
<td>5.6-21</td>
</tr>
<tr>
<td>5.6-10</td>
<td>Control Configuration Byte</td>
<td>5.6-22</td>
</tr>
<tr>
<td>5.6-11</td>
<td>General Q.931 Message Format</td>
<td>5.6-33</td>
</tr>
<tr>
<td>5.6-12</td>
<td>SIMP/L LAPD Mnemonics</td>
<td>5.6-34</td>
</tr>
<tr>
<td>5.6-13</td>
<td>SIMP/L LAPD Configuration Checklist</td>
<td>5.6-40</td>
</tr>
<tr>
<td>5.7-1</td>
<td>SIMP/L LAPD Commands Available in Multi-Link LAPD</td>
<td>5.7-3</td>
</tr>
<tr>
<td>5.7-2</td>
<td>Multi-Link SIMP/L LAPD Commands and Variables</td>
<td>5.7-4</td>
</tr>
<tr>
<td>5.7-3</td>
<td>Multi-Link Frame Status Byte Interpretation (Second Byte)</td>
<td>5.7-6</td>
</tr>
<tr>
<td>5.7-4</td>
<td>Multi-Link SIMP/L LAPD Link States</td>
<td>5.7-11</td>
</tr>
<tr>
<td>5.8-1</td>
<td>V.120 LLI Values</td>
<td>5.8-1</td>
</tr>
</tbody>
</table>

LF-2 • Version 4.7
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8-2</td>
<td>SIMP'L LAPD Commands Available in SIMP'L V.120</td>
<td>5.8-3</td>
</tr>
<tr>
<td>5.8-3</td>
<td>V.120 SIMP'L Commands and Variables</td>
<td>5.8-4</td>
</tr>
<tr>
<td>5.8-4</td>
<td>V.120 Frame Status Byte Interpretation (Second Byte)</td>
<td>5.8-6</td>
</tr>
<tr>
<td>5.8-5</td>
<td>V.120 Link States</td>
<td>5.8-11</td>
</tr>
<tr>
<td>6.3-1</td>
<td>BSC Default Mnemonic Table</td>
<td>6.3-1</td>
</tr>
<tr>
<td>6.4-1</td>
<td>BSC Transmission and Reception Commands</td>
<td>6.4-1</td>
</tr>
<tr>
<td>6.4-2</td>
<td>BSC Miscellaneous Commands</td>
<td>6.4-1</td>
</tr>
<tr>
<td>6.5-1</td>
<td>TXSTATUS Byte Bit Mat</td>
<td>6.5-11</td>
</tr>
<tr>
<td>7.3-1</td>
<td>Async Mnemonic Table</td>
<td>7.3-2</td>
</tr>
<tr>
<td>7.4-1</td>
<td>Async Read-Only Variables</td>
<td>7.4-1</td>
</tr>
<tr>
<td>7.4-2</td>
<td>Async Transmission and Reception Commands</td>
<td>7.4-1</td>
</tr>
<tr>
<td>8.2-1</td>
<td>SITREX Automatic X.25 Simulator Options</td>
<td>8.2-2</td>
</tr>
<tr>
<td>8.4-1</td>
<td>SITREX Trace Commands</td>
<td>8.4-2</td>
</tr>
<tr>
<td>8.6-1</td>
<td>SITREX Alphabetical Command Index</td>
<td>8.6-2</td>
</tr>
<tr>
<td>8.6-2</td>
<td>SITREX Frame Level Commands</td>
<td>8.6-8</td>
</tr>
<tr>
<td>8.6-3</td>
<td>SITREX Packet Level Commands</td>
<td>8.6-8</td>
</tr>
<tr>
<td>8.6-4</td>
<td>SITREX Parameter Commands</td>
<td>8.6-9</td>
</tr>
<tr>
<td>8.6-5</td>
<td>SITREX Wait Commands</td>
<td>8.6-9</td>
</tr>
<tr>
<td>8.6-6</td>
<td>SITREX Loop, Jump, and Reinitialization Commands</td>
<td>8.6-11</td>
</tr>
<tr>
<td>8.6-7</td>
<td>SITREX Program Management Commands</td>
<td>8.6-10</td>
</tr>
<tr>
<td>8.6-8</td>
<td>SITREX Numeric Variable Commands</td>
<td>8.6-10</td>
</tr>
<tr>
<td>8.6-9</td>
<td>SITREX Message Buffer Commands</td>
<td>8.6-11</td>
</tr>
<tr>
<td>8.6-10</td>
<td>SITREX Trace Buffer Commands</td>
<td>8.6-11</td>
</tr>
<tr>
<td>8.6-11</td>
<td>SITREX Display and Print Commands</td>
<td>8.6-11</td>
</tr>
<tr>
<td>8.10-1</td>
<td>SITREX Simulator Reactions</td>
<td>8.10-1</td>
</tr>
<tr>
<td>8.10-2</td>
<td>SITREX Simulator Reactions</td>
<td>8.10-2</td>
</tr>
<tr>
<td>8.10-3</td>
<td>SITREX Simulator Reactions</td>
<td>8.10-3</td>
</tr>
<tr>
<td>8.10-4</td>
<td>SITREX Simulator Reactions</td>
<td>8.10-4</td>
</tr>
</tbody>
</table>
1.1 CHAMELEON SIMULATION CONCEPTS

Why use simulation?
Simulation allows you to emulate either side of the line in an X.25, SNA, Async, BSC, ISDN Q.921/Q.931, or any BOP framed protocol. Using the Chameleon's protocol-specific programming languages, you can establish a testing environment for your specific needs. Chameleon simulation allows you to:

- Create a controlled live environment to test both hardware and software
- Test both common and not-so-common error conditions
- Simulate complex network or individual devices such as:
  - Mainframes or Front End Processors
  - Modems
  - Terminals or Terminal Controllers
  - ISDN Network Terminator (NT) or Terminal Endpoint (TE)
  - Other intelligent communications devices

Beginning with Chameleon 32 system software 4.0 and Chameleon 20 system software 1.2, you can simultaneously monitor the data being transmitted between the Chameleon and the device under test. Refer to Section 2.1 for more information about this capability.

Applications
You can use simulation for the following types of applications:

- Use the simulation languages to write certification packages to test equipment or software functionality by writing protocol-specific scenarios. The Chameleon itself was tested using its own simulators to send and receive worst case data.

- After identifying a specific problem using Chameleon Analysis (SABM collision, for example) you can develop simulation programs to duplicate the problem and continue testing until the hardware or software is able to handle the problem correctly.

- If you are having problems running certification packages (for example, DDN tests), you can copy just the portion of the test package you need, modify it, and run it. This enables you to run a part of the test so that you can focus on that portion without running the entire test package repeatedly.
WHY BASIC?

Introduction

The Chameleon BASIC language is an interpreted test language that looks like standard BASIC. Since BASIC is a widely used language, many Chameleon users are already familiar with its general format and usage and can adapt quickly to Chameleon BASIC and its subsets.

Chameleon BASIC is structured so that you can develop programs quickly, and has the added benefit of providing an interactive interface to the user. You can enter many commands directly at the BASIC prompt, and they will be executed immediately.

The ability to write your programs and run them without compiling means that you can reduce the amount of time required to debug programs.

Chameleon 32
C Development System

There is also a C Development System available for the Chameleon 32. The C package gives you the advantage of working in a familiar C programming environment. It includes special libraries containing protocol-specific functions. These libraries are listed below with their BASIC simulator counterparts:

<table>
<thead>
<tr>
<th>C Library</th>
<th>Comparable to using BASIC:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOP Library</td>
<td>FRAMEM</td>
</tr>
<tr>
<td>HDLC</td>
<td>SIMP/L HDLC</td>
</tr>
<tr>
<td>SDLC</td>
<td>SIMP/L SDLC</td>
</tr>
<tr>
<td>LAPD</td>
<td>SIMP/L LAPD</td>
</tr>
<tr>
<td>ASYNC</td>
<td>Chameleon ASYNC</td>
</tr>
<tr>
<td>BSC</td>
<td>Chameleon BSC</td>
</tr>
<tr>
<td>ISDN Basic Rate Interface</td>
<td>Basic Rate Setup Menu</td>
</tr>
<tr>
<td>ISDN Primary Rate Interface</td>
<td>Primary Rate Setup Menu</td>
</tr>
</tbody>
</table>

BASIC vs. C

BASIC provides a user friendly interface through the use of set-up menus for configuring various parameters. C does not have menus; the programmer must use the library functions to configure parameters.

BASIC has a shorter development cycle. It can be used to develop simpler program very quickly. It is possible use BASIC to determine your fundamental test design, and then develop the complete test system using C.
1.2 SIMULATION LANGUAGES

Introduction

The Chameleon offers you five programming languages with special protocol-specific commands and extensions.

Four of the simulation languages are part of the Chameleon BASIC family. Chameleon BASIC is the language used for BASIC programming tasks, such as file handling and string and numeric functions. Each of the four protocol-specific languages use Chameleon BASIC commands, plus their own special commands for the applications you need.

Once you learn the Chameleon BASIC language, you have only to learn a few extra commands for the protocol-specific tasks you want to perform. Figure 1.2-1 below shows the relationship between the Chameleon languages.

The fifth protocol-specific language, SITREX, is not a part of the Chameleon BASIC family.

Figure 1.2-1: The Chameleon BASIC Family
FRAMEM is the Chameleon’s flexible FRAMe EMulator for the HDLC, SDLC, LAPD, and DMI protocols. FRAMEM includes protocol-specific commands for transmission and reception of these protocols.

SIMP/L is the Chameleon’s Simulated Interactive Multi-Protocol Language for HDLC, SDLC and LAPD. SIMP/L includes automatic support for the link level of these protocols.

Chameleon ASYNC is the Chameleon’s Async Simulation Language. It includes specific commands for asynchronous transmission and reception.

Chameleon BISYNC is the Chameleon’s Bisync Simulation Language. It includes specific commands and variables for binary synchronous transmission and reception.

The fifth protocol specific language, SITREX, provides automatic X.25 simulation for levels 2 and 3. SITREX is not a part of the Chameleon BASIC family.

The table below shows the applications for the protocol-specific simulation languages.

<table>
<thead>
<tr>
<th>PROTOCOL LEVEL SIMULATED</th>
<th>X.25/HDLC</th>
<th>SNA/SDLC</th>
<th>BISYNC</th>
<th>ASYNC</th>
<th>LAPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>FRAMEM HDLC and SITREX</td>
<td>FRAMEM SDLC</td>
<td>CHAMELEON BISYNC</td>
<td>CHAMELEON ASYNC</td>
<td>FRAMEM</td>
</tr>
<tr>
<td>3+</td>
<td>SIMP/L HDLC and SITREX</td>
<td>SIMP/L SDLC</td>
<td></td>
<td></td>
<td>SIMP/L</td>
</tr>
</tbody>
</table>

Table 1.2-1: Simulation Languages
SIMULATOR APPLICATIONS

An example of a test application for each simulator is provided below to give you a better understanding of the function of each.

FRAMEM HDLC/SDLC  Test the link establishment connection.
FRAMEM DMI  Test Mode 2 handshaking for DMI.
FRAMEM LAPD  Test TEI assignments for Q.921.
SIMP/L HDLC  Develop a call generator that loads the switch with active X.25 calls.
SIMP/L SDLC  Test a particular type of BIND for SNA.
SIMP/L LAPD  Test call setup procedures for Q.931.
BSC  Test poll times between a cluster controller and terminals for BSC.
ASYNC  Test printer or terminal setups.
SITREX  You need full X.25 simulation to place a call to determine if the X.25 line is active.
FILE NAME EXTENSIONS

When you create a simulation program, a file name extension is automatically assigned to the program file, using the conventions shown in the table below. Do not modify the file extension or you will be unable to run the program from the simulator.

<table>
<thead>
<tr>
<th>FILE TYPE</th>
<th>DIRECTORY</th>
<th>FILE EXTENSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAMEM SDLC/HDLC</td>
<td>\TEKELEC\SIMULATE\FBOP</td>
<td>.CB</td>
</tr>
<tr>
<td>FRAMEM LAPD</td>
<td>\TEKELEC\SIMULATE\FLAPD</td>
<td>.LB</td>
</tr>
<tr>
<td>FRAMEM DMI</td>
<td>\TEKELEC\SIMULATE\FDMDI</td>
<td>.JB</td>
</tr>
<tr>
<td>SIMP/L HDLC</td>
<td>\TEKELEC\SIMULATE\SHDLC</td>
<td>.EB</td>
</tr>
<tr>
<td>SIMP/L SDLC</td>
<td>\TEKELEC\SIMULATE\SSDLC</td>
<td>.DB</td>
</tr>
<tr>
<td>SIMP/L LAPD</td>
<td>\TEKELEC\SIMULATE\SLAPD</td>
<td>.MB</td>
</tr>
<tr>
<td>BISYNC</td>
<td>\TEKELEC\SIMULATE\BISYNC</td>
<td>.HB</td>
</tr>
<tr>
<td>ASYNC</td>
<td>\TEKELEC\SIMULATE\ASYNC</td>
<td>.IB</td>
</tr>
<tr>
<td>SITREX</td>
<td>\TEKELEC\SIMULATE\SITREX</td>
<td>.BA</td>
</tr>
</tbody>
</table>

Table 1.2-2: Simulation Directories and File Extensions
2.1 STARTING SIMULATION

Introduction

This chapter describes configuring the Chameleon for simulation. If you are unfamiliar with the Chameleon, you should first read the Chameleon User's Guide, Chapters 1 and 3 before you read this chapter.

To configure the Chameleon for Simulation, do the following:

1. Power up and boot the Chameleon.

2. After the system software is booted, a Configuration menu appears (Figure 2.1-1).

   If this menu is not displayed, the arrow cursor will be positioned at the Setup Mode parameter. To display the menu, press F1 Menu.

3. Move the arrow cursor to the Mode of Operation parameter and press F2 Simulate.

Figure 2.1-1: Menu Setup Mode (Single Port)
4. Use the function keys to select the appropriate **Physical Interface** for your application.

If your machine includes a Basic Rate or a Primary Rate Interface, you will have an **F7 Physicl** softkey, which displays a menu for configuring the physical interface. Refer to the following for more information:

- For *Primary Rate Interface*, refer to the *Chameleon Protocol Interpretation Manual*, Chapter 11.
- For *Basic Rate Interface*, refer to the *Chameleon Protocol Interpretation Manual*, Chapter 12.

5. Press **F6 Setup** to display the menu of available protocols (Figure 2.1-2).

6. Press the function key that corresponds to the protocol you want to simulate. Use the **F9 More** key to display additional protocol options, if needed.

![Figure 2.1-2: Chameleon Protocol Setup Menu](image-url)
7. Each protocol has additional parameters. Press Go to accept the default values, or modify the parameters, and then press Go. This returns you to the main configuration menu.

(To modify a parameter, move the arrow cursor to the parameter and press the desired function key. These parameters are described in the Chameleon Protocol Interpretation Manual, Volume II.)

8. Press Go and the Applications Selection Menu is displayed (Figure 2.1-3). This menu displays the applications that are available for the selected protocol. There are two windows:

- Simulation, which displays the simulators (languages) available for the selected protocol, and are designated as follows:

  SM_HDLC  SIMPL HDLC
  SM_SDL C  SIMPL SDLC
  SM_LAPD  SIMPL LAPD
  SM_MLAPD Multi-Link SIMPL LAPD
  SM_V120  SIMPL V.120
  ASYNC     ASYNC
  BISYNC    BISYNC
  FR_HDLC  FRAMEM HDLC
  FR_SDL C  FRAMEM SDLC
  FR_DMI   FRAMEM DMI
  FR_LAPD  FRAMEM LAPD
  STREX    STREX
  T_STREX  TRANSPAC STREX

- Monitoring, which enables you to analyze the data being transmitted between the Chameleon and the device under test during simulation.

9. To load one of the simulation applications, do the following:

a. Press Shift ↓ to Select the Simulation window.

  - On the Chameleon 20, the selected window contains the blinking arrow cursor. The inactive window has a non-blinking arrow cursor.

  - On the Chameleon 32, the selected window contains the red arrow cursor. The inactive window contains a white arrow cursor.

b. Move the arrow cursor to the desired simulator.
Chameleon BASIC Simulation Manual

Ch. 2.1: Starting Simulation

Figure 2.1-3: Applications Selection Menu

**Up/Down Arrow:** Indicates which arrow key to press to display additional applications in that window.

**Arrow cursor:**
- Red arrow cursor indicates active window.
- White arrow cursor indicates inactive window.
- Press Shift ↑ or Shift ↓ to change active window.

**Reset:**
- Resets all applications, for example:
  - Clears History buffer.
  - Resets all statistics values to 0.

**Menu:**
- Displays the first configuration menu without stopping applications.

**Save:**
- Saves the displayed configuration to a named file.

**Simulation Window:**
- Displayed in Simulation mode only.
- Selects a Simulator, for example:
  - SM_HDLC = Simp1 HDLC

**Exit:**
- Stops all applications and returns to the first configuration menu.

**Set T.O:**
- Sets the value of the autoboot timer.

**Monitoring Window:**
- Displayed in Monitoring and in Simulation modes.
c. Press the function key which loads the simulator on the desired port.

While the application is loading, the port letter (A or B) appears blinking next to the application name. When loaded, the letter stops blinking.

10. To load Monitoring applications, select the Monitoring window and repeat the procedure in step 9. You can run several Monitoring applications on each port.

11. When all applications are loaded, press Go. This starts the applications, as indicated by the appearance of page banners at the bottom of the screen.

12. You are now ready to use the applications that are running. The Simulate page accesses the Simulator you loaded.

On a Dual Port machine, if Simulators are running on both ports, the page banner display Simulate A or Simulate B to indicate the port.

The other pages are Monitoring applications and are described in the Chameleon User’s Guide.

13. Select and display the Simulate page. To do this, press the Select key until the Simulate banner is highlighted. Then use one of the keys below to display the page.

<table>
<thead>
<tr>
<th>KEY</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move ↑</td>
<td>Moves the page banner up one line at a time (increases the page size).</td>
</tr>
<tr>
<td>Move ↓</td>
<td>Moves the page banner down one line at a time (decreases the page size ).</td>
</tr>
<tr>
<td>Scroll ↑</td>
<td>Scrolls the data displayed in the page upward one line at a time.</td>
</tr>
<tr>
<td>Scroll ↓</td>
<td>Scrolls the data displayed in the page downward one line at a time.</td>
</tr>
<tr>
<td>Shift Scroll ↑</td>
<td>Scrolls the data in the page up the number of lines displayed in the page.</td>
</tr>
<tr>
<td>Shift Scroll ↓</td>
<td>Scrolls the data in the page down the number of lines displayed in the page.</td>
</tr>
<tr>
<td>Shift Hide Page</td>
<td>Hides the active page so that the banner is no longer visible on the screen (the application continues to run).</td>
</tr>
<tr>
<td>Show Page</td>
<td>Displays a page that has been hidden with Shift Hide Page.</td>
</tr>
<tr>
<td>Replace</td>
<td>Replaces the active page with one that has been hidden using Hide Page.</td>
</tr>
<tr>
<td>Shift Move ↑</td>
<td>Displays the page in a special full-screen mode referred to as Blow Mode (indicated by the letter B on the top left side of the banner). Other pages cannot be accessed when the active page is in Blow Mode. Shift Move ↑ again disables Blow Mode, and returns the screen to its previous state.</td>
</tr>
</tbody>
</table>

Figure 2.1-4: Page Manipulation Keys
14. For **BASIC** simulators, the Simulation ! prompt is displayed. At the simulation prompt (!), you can enter commands and run programs.

A protocol-specific parameter file named DEFAULT is automatically loaded for **BASIC** simulators. You can view and/or modify the parameters by entering the **BASIC** command:

\[ \text{! setup } \text{<Return>} \]

For **SITREX** or **T.SITREX**, a parameter setup menu is displayed. There are three menus of configuration parameters for **SITREX**. To use the default values that are displayed, press Z and you are taken into the simulator.

The **setup** command is not available in **SITREX**. For this reason, you are taken to the setup menus before you access the simulator. This gives you the opportunity to modify the setup, or load a different parameter setup file, before you begin your application.

Parameter setup menus are described in detail in Section 2.2. The specific parameters for each menu are described in the appropriate chapter of the manual. For example, the FRAMEM HDLC parameters (Figure 2.1-5) are described in Chapter 4.4.

---

**FRAMEM PARAMETER SET-UP MENU**

- **A.** Simulate ........ : DCE
- **B.** Data Encoding .... : NRZ
- **C.** Coupler ........... : HDLC/SDLC
- **D.** Bit Rate .......... : 56000

**SELECT >**

**CHOICES:**

- Z = run FRAMEM,
- ? = Help,
- S = Save parameters,
- ESC = Main Menu

**Figure 2.1-5: FRAMEM Parameter Set-Up Menu**
15. To change your port configuration (protocol, simulation language, physical interface) you must first stop the Simulator on the port, as follows:

- For BASIC simulators, you can stop the Simulator and return to the port configuration menu by entering the following command at the ! prompt:

  \[! \text{ menu} \] \(<\text{Return}>\]

- Alternately, for all simulators, you can select the Configuration page, which displays the Applications Selection menu, and press one of the following:
  
  \(\triangleright\) Press \(F10 \text{ Exit}\) to stop all applications that are currently running on all ports.

  \(\triangleright\) In the Simulation window, move the cursor to the Simulator, and press the softkey that stops it on the desired port (\(F1 \text{ Stop A, F2 Stop B or F3 Stop AB}\)).

If you access the first configuration menu, and the Simulator is still running, the message \textit{Busy} appears next to the port parameters. When this message appears, you cannot change any of the port configuration parameters for that port.

16. Figure 2.1-6 summarizes the steps required to move between the simulation prompt (!), the setup menus, and the main configuration menus.
### Table 2.1-6: Simulation Next Steps

<table>
<thead>
<tr>
<th>If you are here:</th>
<th>NEXT STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAMEM, SIMP/L, BSC, or Async simulation prompt (!)</td>
<td><strong>And you want to:</strong></td>
</tr>
<tr>
<td>Return to the Applications Selection menu</td>
<td>Enter: MENU &lt;RETURN&gt;</td>
</tr>
<tr>
<td>Access the parameter set-up menu</td>
<td>Enter: SETUP &lt;RETURN&gt;</td>
</tr>
<tr>
<td>Write programs</td>
<td>Read Chapter 3 for Chameleon BASIC fundamentals AND read the chapter that describes your simulation language.</td>
</tr>
<tr>
<td>SITREX SIMULATOR ACTIVE</td>
<td>Return to the port configuration menu</td>
</tr>
<tr>
<td>Access the parameter set-up menu</td>
<td>Enter: PS &lt;RETURN&gt; HALT &lt;RETURN&gt;</td>
</tr>
<tr>
<td>Press: F2 GO</td>
<td>Enter: PS &lt;RETURN&gt;</td>
</tr>
<tr>
<td>Enter command mode (!)</td>
<td>Enter: EXIT &lt;RETURN&gt;</td>
</tr>
<tr>
<td>Exit command mode</td>
<td>Enter: PP &lt;RETURN&gt;</td>
</tr>
<tr>
<td>Activate the trace buffer</td>
<td>Enter: CTRL P</td>
</tr>
<tr>
<td>Deactivate the trace buffer</td>
<td>Access the simulation prompt (!) to write programs Press: Z</td>
</tr>
<tr>
<td>Any Parameter Set-Up menu</td>
<td>Return to the port configuration menu Press: ESC</td>
</tr>
<tr>
<td>Change parameter values</td>
<td>Read Chapter 2.2 for general information about the set-up menus AND read the chapter that describes the menu for the simulation language you are using.</td>
</tr>
<tr>
<td>Access the Save Parameters menu</td>
<td>Press: S</td>
</tr>
</tbody>
</table>
2.2 PARAMETER SET-UP MENUS

Introduction

The parameter set-up menus enable you to configure protocol-specific parameters and initialize the Chameleon Front End Processor (FEP) for simulation.

Before you can transmit or receive data during simulation, you must initialize the Chameleon FEP. This is done when a parameter set-up file is loaded.

Each simulator is provided with a parameter setup file named DEFAULT. You can modify the DEFAULT file or create additional setup files with user-specified names. The ability to save more than one set-up file enables you to quickly load and set up the parameters for a given application. (See Save Parameters, Section 2.3.)

When you start a simulator, you are taken directly to the simulation prompt !. If you have a parameter set-up file for that Simulator named DEFAULT, it is automatically loaded, and the FEP is initialized with the values in that file.

If you attempt to transmit traffic and a parameter set-up file has not been loaded, the following message is displayed:

P1 SETUP NOT PERFORMED

This should only occur if you delete the DEFAULT file. If this message is displayed when using a simulator, you can do the following:

a. At the ! prompt, type: SETUP to display the menu.
   b. Change the settings as needed.
   c. Press Z to exit the menu and return to the ! prompt

General instructions for using a parameter set-up menu begin on the following page. The menus are described in the protocol-specific sections of the manual.
Using Set-Up Menus

In this section, FRAMEM HDLC will be used as an example to describe the use of the parameter set-up menu. The FRAMEM HDLC Parameter Set-Up Menu is shown in Figure 2.2-1, with the original default values displayed. The menu contains four parameter options (A - D).

![FRAMEM Parameter Set-Up Menu](image)

Figure 2.2-1: The FRAMEM Parameter Set-Up Menu

To change a parameter, follow these steps:

1. Type the menu letter (A - D) that corresponds to the parameter you want to change.

   The parameter name is displayed after SELECT on the lower left half of the screen. Acceptable values are listed after CHOICES on the lower right half of the screen. The figure below shows how the screen would look if you selected the A. - Simulate option from the menu. The parameter and the current choice are displayed in reverse video.

   If the parameter requires a numeric value, a message describing the required input is displayed. For these parameters, you are not given a range of valid options. Enter a value within the specified range.
2. Press the *space bar* to move the highlight to the new choice. Each time you press the space bar the highlight moves to the next choice on the list.

3. When the desired choice is highlighted, press RETURN. You can then select another parameter by entering the menu letter.

   Optionally, when the desired choice is highlighted, you can use the *up* and *down* arrow keys to select another parameter. When you use the arrow keys, the new value for the current parameter is automatically entered before the new parameter is selected.
Other Options

The bottom of the parameter set-up page lists other keys with special functions. A description of each option is provided below.

Z = Run {simulator}  
Option Z passes control to the simulator and displays the ! prompt for entering direct commands and program code. The currently displayed parameters are saved automatically in a file called DEFAULT. This file is executed automatically, unless you designate a different set-up file using the COUPLER command at the ! prompt.

? = Help  
Once a parameter is selected, press the ? key to display a definition of the current parameter. A help message is displayed on the lower half of the screen. Some messages may be displayed in two or more parts. Press C to display the next section. Press any other key to leave help.

ESC = Main Menu  
The ESC key returns control to the Chameleon main menu. From the main menu, you can select a different simulator or switch to the Analysis mode.

S = Save Parameter  
Option S takes you to the Save Parameters Menu, which is described in the Section 2.3. This menu enables you to load other parameter set-up files, or save the current settings in a new file.
2.3 SAVE PARAMETERS MENU

The Save Parameters Menu can be accessed from any of the parameter set-up menus. This menu enables you to save one or more parameter set-up files on either the hard or floppy disk drive. By doing this, you do not have to re-enter your settings each time you run your simulation programs; you simply load the desired file from the disk.

Note that you can also use the COUPLER command at the prompt to load and execute a parameter set-up file.

Each of the Save Parameter options is described below.

A - Save as Default file
B - Save as a named file
C - List existing file names
D - Return to SETUP Menu
E - Load a file into SETUP MENU
Z - Run (simulator)
ESC - Exit to Main Menu

Figure 2.3-1: Save Parameters Menu

A - Save as Default File

Option A saves the parameters in a file named DEFAULT on the hard disk drive (A). DEFAULT is the parameter file that is automatically used, unless option E, Load a file into SETUP MENU, is selected.
B - Save as a Named File
Option B saves the parameters in a specified file. When you press B, the prompt asks for the drive specification (A for the hard disk drive or B for the floppy disk drive) and the filename. Filenames must be between one and eight alphanumeric characters.

C - List Existing File Names
Option C displays a list of the existing parameter files on a specified disk drive. Use the down arrow key to display more filenames. Use the up arrow key to display the previously listed filenames.

D - Return to SETUP Menu
Option D returns you to the parameter set-up menu.

E - Load a File into SETUP Menu
Option E loads a parameter set-up file from disk into the set-up menu. Loading files into the set-up menu is not related to executing them with the COUPLER command, although the setup will be passed to the simulator through the DEFAULT file.

Z - Run {simulator}
Option Z runs the simulator and displays the simulation prompt (!). At the ! prompt you can enter commands and run programs. This command automatically saves the parameters in the DEFAULT parameter file.

ESC - Exit to Main Menu
The ESC key returns you to the main configuration page. From this page you can select a protocol, simulation language, or the Analysis mode.
3.1 CHAMELEON BASIC INTRODUCTION

Introduction

This section introduces some of the features of Chameleon BASIC. It also references sections in the manual that provide more information about specific features.

Commands

In general, Chameleon BASIC commands enable you to:

- Create programs
- Execute programs
- Save files, traffic, and programs

Program Development

An editor allows you to enter, edit and display your program. There is a debug mode which displays line numbers during execution to check loops and GOTO statements. Refer to Section 3.4 for a list of these commands.

File Management

File management commands save programs to disk, load programs from disk to memory, display disk directories, and delete files from disk. Refer to Section 3.4 for a list of these commands.

Data Files

You can also create and control disk-based data files. Refer to Section 3.4 for a list of these commands.

Program Control and Subroutines

Program execution can be started at specific line numbers or subroutines. You can chain programs together for sequential execution or nest them to be called as subroutines. You can also call a program from a disk as a subroutine. Sixteen levels of calling are supported.

Refer to Section 3.4 for a list of these commands.

Variables

Chameleon BASIC has numeric variables, string variables and read-only variables for you to use. You can assign and enter strings and numeric variables from the keyboard, including assigning and printing variables in hex. Refer to Sections 3.2 and 3.4 for more information about variables.
Printing

There are a number of BASIC commands that enable you to print programs, messages, strings, and numeric values. Refer to Section 3.4 for a list of these commands. In addition, you can use the Print Page/Print Scrn key on the Chameleon keyboard.

Refer to the Volume I of the User’s Manual for information about configuring the Chameleon to output to your printer.

String Functions

Chameleon BASIC provides you with code conversion string functions that convert from:

- ASCII to EBCDIC
- ASCII to numeric
- ASCII to BCD
- EBCDIC to ASCII
- ASCII to hexadecimal

Refer to Section 3.4 for a list of these commands.

Timers

A realtime clock provides time stamps and can be accessed using special commands. There are four Chameleon BASIC timers: two that count up and two that count down. Refer to Section 3.2 for more information about timers.

Programming Memory

There are eight Kbytes of user program memory. The read-only variable SIZE displays the amount of programmable memory available.

Video Effects

You can create custom video screen formats within programs using the video effects commands. Refer to Section 3.4 for a list of these commands.

Program Logic

You can perform logical manipulation of data with Boolean operators such as:

- AND
- OR
- XOR (exclusive or)
- # (not)

Refer to Section 3.2 for more information about arithmetic and logical operators.
Mnemonics  Chameleon BASIC includes built-in tables of standard mnemonics for communications. You can also define and save your own mnemonic tables. Refer to Section 3.2 for more information about mnemonic commands.

Trace Buffer  A record of each frame transmitted and received is stored in a special buffer called the trace. You can view the trace at any time. Refer to Section 3.2 for more information about simulation buffers.
3.2 BASIC FEATURES

Introduction

This section describes the major features of Chameleon BASIC in greater detail. The features that are described are:

- Simulation Buffers
- Mnemonics
- Timers
- Arithmetic and Logical Operators
- Variables
- Arrays
SIMULATION BUFFERS

Introduction

There are several buffers that are used to store the traffic that is transmitted and received during simulation. These buffers are:

- Acquisition buffer
- Trace buffer
- DISPF buffer

Acquisition Buffer

The Acquisition buffer stores the data received from the line. This is where the traffic is first stored and uses a FIFO (first-in, first-out) system.

You do not have direct access to the data in the acquisition buffer, although you can clear it using the FLUSH command.

If the acquisition buffer becomes full, the message:

```
   P1 OVERFLOW HIT ANY KEY TO RESET
```

is displayed. When you press a key, the buffer is flushed automatically. To display the data in the acquisition buffer, it must be transferred to the trace buffer, as described below.

Trace Buffer

When you enter a REC command, data in the acquisition buffer is transferred to the trace buffer and the frame length, status (CRC), and a timestamp are added to each frame.

Data is moved from the acquisition buffer to the trace buffer only when there is sufficient space available in the trace buffer. (To determine the amount of space available in the trace buffer, use the TFREE variable.)

Once data is in the trace buffer, you can use the trace buffer commands to access the data. These commands give you the ability to:

- Display the information in the trace (TPRINT)
- Save the information in the trace to a trace file (TSAVE)
- Print the trace information to the printer (LTPRINT)
- Load a trace file into memory (TLOAD)
- Clear all data from the trace buffer (CLEAR)
For example the TPRINT command displays the contents of the trace buffer. In addition to the data the trace buffer display also includes a timestamp, frame length, and status (CRC).

The general format of a trace buffer display is shown below. The display following the frame length is protocol-specific. Refer to the appropriate protocol chapter for more information.

**DISPF Buffer**

The DISPF buffer stores the last frame that was transmitted or received. Following a REC command, the DISPF buffer is emptied. If a frame is received, it is put into the buffer. The length of the buffer varies with the protocol being used.

The contents of the DISPF buffer is displayed using the DISPF command. The contents can be output to a printer using LDISPF.

In SIMP/L there are two DISPF buffers. The RDISPF buffer contains the last frame received and the TDISPF buffer contains the last frame built for transmission. To display the contents of these buffers use RDISPF and TDISPF, respectively.
MNEMONICS

Introduction

A mnemonic is a named variable and/or constant that is used to refer to protocol elements. Each simulator includes a table of standard mnemonics and values that you can use without modification. Instead of defining variables in your program for the various protocol elements, you can use the pre-defined mnemonics in the default table.

The default mnemonic tables are described in each of the protocol-specific chapters.

Mnemonic Commands

You can also develop custom mnemonic tables for specific applications using the BASIC mnemonic commands. The mnemonic commands allow you to:

- Define new mnemonics or redefine existing mnemonics (DEFINE)
- List the entries in the current mnemonic table (MLIST)
- Load a specific mnemonic table file from disk (MLOAD)
- Save a user-defined mnemonic table (MSAVE)

Implementation

There is some variation in the way that the mnemonic tables have been implemented in the BASIC simulators.

In FRAMESM, BSC, and Async mnemonics are used as constant values.

In SIMP/L mnemonics are used as variables and are associated with a field length.

Application Note

Heavy use of mnemonics causes the simulator to slow down because of the time required for mnemonics to be looked up in the table. A mnemonic table may contain a maximum of 255 mnemonic entries; however, to maximize the speed of the simulator, it is recommended that you restrict your mnemonic tables to approximately 20 entries.

If you have developed a program that uses many mnemonics and you want to increase the speed of its execution, you may want to replace your mnemonics with variables.
CHAMELEON BASIC TIMERS

There are four timers in Chameleon BASIC. You can use these timers anywhere a normal variable is valid. The four timers are:

- TIM0  Counts down in ten millisecond (.01) intervals
- TIM1  Counts up in ten millisecond (.01) intervals
- TIM2  Counts down in seconds
- TIM3  Counts up in seconds

Assignment

Timers can be assigned a starting value. For example:

```
TIM2 = 120
TIM1 = 1000
```

Terminal Values

The timers stop when they reach their terminal values. Timers TIM0 and TIM2 (which count down) have terminal values of zero. TIM1 and TIM3 (which count up) have terminal values of FFFF in hex or -1 in decimal.

Example

The following sample program uses the TIM2 timer to test receiving a transmission from the line.

```
10 CLEAR
20 TIM2 = 120
30 REC
40 IF RXFRLEN # 0 GOTO 80
50 IF TIM2 > 0 GOTO 30
60 PRINT "TIMED OUT"
70 STOP
80 REM
```

Clears the trace buffer.
Sets the timer to count down by seconds for two minutes.
Receives a frame from the line.
If the frame length does not equal zero, continues with the program.
If timer has not expired, tries to receive another frame.
Timer expired, prints message.
Terminates program.
Continues processing.
ARITHMETIC AND LOGICAL OPERATORS

Introduction

Arithmetic and logical operators manipulate numeric variables. Operators add, subtract, divide, multiply, etc. The arithmetic and logical operators in Chameleon BASIC can be used as expressions in many commands, such as LET and IF.

Arithmetic Operators

There are four valid arithmetic operators. These are:

- + Add
- - Subtract
- * Multiply
- / Divide

Integers are 16-bit signed integers in the range -32767 through +32768.

Logical Operators

There are 11 logical (Boolean) operators. These operate on 16-bit integer variables and are listed below. A Boolean logical operation that is evaluated as true, returns a value of 1. A Boolean logical operation that is evaluated as false, returns a value of 0.

The Boolean logical operators are:

- = Equal
- # Not equal
- > Greater than
- >= Greater than or equal
- < Less than
- <= Less than or equal

The Bitwise logical operators are:

- AND Logical AND
- OR Logical OR
- XOR Logical exclusive OR
- SHLxx Shift left xx bits (maximum 15)
- SHRxx Shift right xx bits (maximum 15)

Example: In this example, A is set equal to the hex value of 10 shifted right by two places.

A = 16 SHR02

This returns a value of 4.
Precedence of Operators

The order of precedence of arithmetic operators is:

* , /  Multiplication and division
+ , -  Addition and subtraction

To change the order in which the operations are performed, use parentheses. Operations within parentheses are performed first. Inside parentheses, the usual order of operations is maintained.

When arithmetic and relational operators are combined in one expression, the arithmetic is always performed first.

The order of precedence of logical operators is:

AND
OR
XOR

Logical operations are performed after arithmetic and relational operations.

Examples

The following examples demonstrate the way Chameleon BASIC handles precedence of operators. The two examples have different results because of the way they are grouped and the way the arithmetic operators are given priority.

Example 1:  
10 LET A = 1  
20 B = 1 + 2 > A

In Example 1, B returns a value of 1. The arithmetic operation 1 + 2 has precedence over the logical operation >. Since 1 + 2 = 3, and 3 is greater than the value of A (1), the statement is true. Since the result is true, the value of B becomes 1.

Example 2:  
LET A = 1  
B = 1 + (2 > A)

In Example 2, B returns a value of 2 because parentheses are used to change the precedence of operators. The parentheses around the operation (2 > A) give the logical operator precedence, and it is evaluated first.

Since 2 is greater than the value of A (1), the operation is true and results in a value of 1. Now the arithmetic operation 1 + 1 is performed, resulting in a value of 2 for B.
VARIABLES

Introduction This section covers the three types of variables that you can use in Chameleon BASIC:

- Numeric variables
- String variables
- Read-Only variables

NUMERIC VARIABLES

Numeric variables have the following characteristics:

- 16-bit signed quantities
- Range: -32767 to 32768
- Can be represented in hex or decimal
  - An ampersand (&) before the number indicates that
  number is in hex representation. For example &10 is hex
  10
- Allowable variable names: A - Z

Numeric variables in Chameleon BASIC are referred to by a single character in the range A to Z. Arithmetic is integer only; use of decimal points results in incorrect results or syntax errors.

You can assign values to numeric variables in either decimal or hexadecimal. Decimal values are acceptable in the range from -32768 to 32767 and are integer only.

Hex Values

Hexadecimal values may range between zero and FFFF. Note that 8000 hexadecimal is equal to -32768 in decimal, and FFFF hexadecimal is equivalent to -1 in decimal. To assign a hexadecimal value to a variable, precede the value with the ampersand symbol (&).

Example: To assign variable Z a hexadecimal value of 1A, enter:

\[ Z = \&1A \]

This is equivalent to 26 in decimal.
STRING VARIABLES

Introduction

Chameleon BASIC string variables have the following characteristics:

- Strings of characters of ASCII
- Characters are 8 bits
- One dimensional arrays
- Allowable variable names are $A - $Z

You can use 26 string variables in Chameleon BASIC. As with the numeric variables, you use characters in the range A - Z for the variable name, preceded by the dollar sign symbol ($). For example: $A, $B, $C.

Assigning Values

The value you assign to a string variable must be enclosed in double quotation marks (" "). The maximum length of a string is 255 characters.

Example: To assign a value of ABCDEF to the variable $A, enter:

$A = "ABCDEF"

String Commands

There are a number of Chameleon BASIC string commands that convert and manipulate strings. These are listed in Section 3.4.

String commands cannot be used directly inside an IF statement as demonstrated in the example below:

INCORRECT: 10 IF MIDS($A,2,1) # MIDS ($B,2,1) GOTO 1000

CORRECT: 10 $A = MIDS ($A,2,1)
20 $B = MIDS ($B,2,1)
30 IF $B # $A GOTO 1000
READ-ONLY VARIABLES

There are five read-only variables in Chameleon BASIC, as described briefly in the figure below. For a complete description of each read-only variable, refer to Section 3.5. There are also protocol-specific read-only variables, which are described in the relevant chapters.

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF</td>
<td>Indicates an end-of-file condition has been detected in an input file.</td>
</tr>
<tr>
<td>FREE</td>
<td>Returns the number of free mnemonic table entries.</td>
</tr>
<tr>
<td>SIZE</td>
<td>Returns the size of free program area in bytes.</td>
</tr>
<tr>
<td>TFREE</td>
<td>Returns the length of the unused trace buffer in bytes.</td>
</tr>
<tr>
<td>TIME</td>
<td>Returns one byte of real time, in specified unit.</td>
</tr>
</tbody>
</table>
ARRAYS

Chameleon Basic has one array available. It is referenced using the @ symbol. The maximum number of values allowed is 256. Subscripts start at zero. Like timers, array elements can be used anywhere that normal variables are valid. It uses the syntax:

- @\(\text{exp}\)

where: @ is the symbol for the array, and \text{exp} is any valid expression that will be used as a subscript in the array.

Example 1: To fill the array with decreasing values, you could enter:

\begin{verbatim}
10 FOR A=0 TO 255
20 @A=255-A
30 NEXT A
\end{verbatim}

Example 2: To access elements in the array, you use the subscript to assign the element of the array to a variable, as shown below:

\begin{verbatim}
B = @8
\end{verbatim}

Example 3: To print the elements of the array, use the PRINT command as in the example below:

\begin{verbatim}
10 FOR A=0 TO 255
20 PRINT "Array(";A;")"=":@A"
30 NEXT A
\end{verbatim}
3.3 INTRODUCTION TO BASIC PROGRAMMING

Introduction

This section describes the fundamentals of BASIC programming. If you are an experienced BASIC programmer, you can skip this section and continue with Section 3.4.

A program is a sequence of logically-ordered statements that are stored in a file. For example, a program can cause the Chameleon to act as a specific type of device to test the effectiveness of communications hardware or software you are developing.

To write a program, you must be familiar with the Chameleon BASIC commands, general syntax rules, and the Chameleon editor. This section provides that information.

Command Types

In general, Chameleon BASIC commands enable you to create programs, execute programs, save files, traffic, and programs. There are three different types of BASIC commands:

- Direct commands
- Statement commands
- Commands that are used as either Direct or Statement

Direct commands are entered at the ! prompt and do not have line numbers. They are executed immediately after you press the RETURN key and therefore are not used within programs. Generally, direct commands enable you to:

- Enter programs
- Load program files from disk
- Save programs to disk
- Debug programs

Some examples of direct commands are:

- AUTO Provides automatic line numbering
- LOAD Loads a program from disk
- RUN Runs a program
- SAVE Saves a program to disk

When entering commands interactively at the ! prompt, there is a history buffer that stores the last command line that you entered. You can restore the line in the buffer to the ! input line by pressing CTRL Q. This enables you to correct syntax errors more quickly.
Statement commands have line numbers and are used in programs. If a statement command is entered at the ! prompt, a Syntax error message is displayed.

Statement commands are combined with other commands to create BASIC programs. For example:

- Assignment
- Arithmetic
- Program flow control

A program is simply a logically-ordered sequence of statement commands that are saved in a file.

Two examples of Statement commands are:

- GOTO x  Executes the command on line x.
- RETURN  Returns to a specific location in a program

Many BASIC commands can be used as Direct or Statement commands and can be entered either in programs or at the ! prompt. Often these commands are most useful when used in programs, since they often depend on other commands to be effective.

For example, the PRINT command, which displays text, values or strings on the Chameleon screen, can be used as either a statement or direct command.

The command descriptions in Section 3.5 indicate whether the command is used as Direct, Statement, or either.

General Syntax Rules

Follow these guidelines for using direct and statement commands:

- Commands can be entered in upper or lower case letters
- Press RETURN to enter a command
- Line numbers are in the range 1 - 9999
- A blank space follows a line number
Command Line Editing

When entering commands interactively at the ! prompt, there is a history buffer that stores the last command line that you entered. You can restore the line in the buffer to the ! input line by pressing CTRL Q. This enables you to correct syntax errors more quickly.

Exiting from the ! Prompt

You can exit from the ! prompt, by entering one of the commands below:

- MENU (Displays the main configuration page)
- SETUP (Displays the parameter set-up menu)

Abort Program Execution

To stop a program while it is running, press the ESC key. This immediately stops the program and returns you to the ! prompt.

Pause Program Execution

If a program is running, you can pause program execution by pressing CTRL S. To continue program execution, press any key.

Abort Program Execution

If a program is running, but has paused to wait for input, you can abort the program by pressing CTRL C or CTRL Z. A program pauses for input as a result of an INPUT, READ, or $INPUT command. Refer to these commands for more information.

Autoexecuting Programs

If you want to have a program that executes automatically, give it the file name DEFAULT. When the simulator is started, the file will automatically load and execute.
COMMAND ABBREVIATIONS

One of the special features of Chameleon BASIC is that you can abbreviate commands, thus shortening the time it takes to write and execute programs.

In general, commands are abbreviated by entering enough letters to identify the command uniquely, and typing a period (.) at the end of the word.

For example, you can enter the AUTO command as:
- AUTO
- AUT.
- AU.
- A.

Since there are no other commands that can be abbreviated that begin with the letter A, the single letter A uniquely identifies the AUTO command. Note that the abbreviation must end with a period.

Use caution when abbreviating similar commands, such as LLIST and LIST. Acceptable abbreviations are:
- LLI. (for LLIST)
- Ll. (for LIST)

Since they both begin with the letter L, you cannot use L. as the abbreviation since this would be ambiguous.

Abbreviations for each command are listed as part of the command description in Section 3.5.

Note: Abbreviations are subject to change without notice when new commands are added to the simulators.

Be aware that the use of abbreviations in a program can make a program difficult to read. However, it is helpful to use abbreviations in the following circumstances:

- You are using commands as direct commands at the \! prompt. The use of abbreviations enables you to type fewer characters to achieve the same functions.

- The program you are writing is getting too large for the amount of programming memory available. Using abbreviations decreases the number of characters in the program file, and therefore decreases the amount of memory used.
COMMAND ERRORS

When an error is encountered in a program, or when entering commands at the ! prompt, the command line is displayed with an arrow pointing to the location of the error. For example, if the CLS (clear screen command) were entered as CSL, the following message appears:

C→SL

When entering commands interactively at the ! prompt, there is a history buffer that stores the last command line that you entered. You can restore the line in the buffer to the ! input line by pressing CTRL Q. This enables you to correct syntax errors more quickly.

Another common error message is:

FILE NOT FOUND

This indicates that you attempted to access a file that could not be located on the indicated disk drive. If this message occurs, check the spelling of the file name and make sure that you are referencing the correct drive. (A is the hard drive and B is the floppy drive.) You can use the FILES command or the File Management menu to list your files.
BRIEF HANDS-ON TUTORIAL

The following is a hands-on tutorial to give you practice entering direct and statement commands. If you want more information about any of the commands that are used, look the command up in the alphabetical listing in Section 3.5.

You can write a simple program, following these steps:

1. Make sure you are at one of the BASIC simulation prompts. (If you don't know how to get to a simulation prompt, read Section 2.1, Selecting a Simulator.)

2. At the ! prompt, type:
   
   `!auto < RETURN>`

   This is the direct command that provides automatic line numbering for you. It begins at line number 10 and increments each line number by 10.

3. Enter the next two commands:
   
   ```
   10 PRINT "HELLO"
   20 GOTO 10
   ```

   When this program is executed, the PRINT command will print the text between the quotation marks to the screen. The GOTO command will send control back to line 10 to repeat the HELLO message forever.

4. At line 30, press RETURN. This exits the auto numbering mode and returns you to the ! prompt.

5. To execute the program, type:
   
   `!RUN < RETURN>`

   The RUN command executes the program that is currently in memory. The word “HELLO” should be displayed repeatedly on the Chameleon screen.

6. To stop the program, press the ESCape key. Generally, you include the STOP command in your program to stop program execution.

7. After writing a program, you should save it to disk using the SAVE command. If you do not save a program to disk, it is erased when you turn the Chameleon off, load a different program from the disk, or write a new program.
To save this program in a file called SAMPLE on the hard disk, enter:

!SAVE"A:SAMPLE" <RETURN>

8. To see the list of files on the disk, enter:

!FILES A <RETURN>

SAMPLE should be listed as a program file.

9. To erase the current program from memory so that you can write a different program, enter:

!NEW <RETURN>

10. To see the program lines currently in memory, enter:

!LIST <RETURN>

Since the NEW command clears the program in memory, there should be no program lines listed.

11. To load and list the SAMPLE program, enter:

!LOAD"A:SAMPLE"

!LIST

The 2-line program should be displayed on the screen.

12. There is a history buffer which stores the last command line entered at the ! prompt. The command line can be restored to the current ! prompt by using CTRL Q.

Enter, the following remark command, including the misspelled word:

!REM This contains an arrow. <RETURN>

The ! prompt is displayed and the command is stored in the history buffer. To restore the command, press:

CTRL Q
The line is displayed at the ! prompt. Use the left arrow key to delete the word arrow and then replace it with the word error. Press RETURN to enter the command. At the ! prompt, enter:

!REM Hello (do not press RETURN)

To replace the current line with the line in the history buffer, press CTRL Q again. The current command line now reads:

!REM This contains an error.
3.4 BASIC COMMAND INDEX

Introduction

This section lists, by function, the commands that are common to all the Chameleon BASIC languages: FRAMEM, SIMP/L, BSC and ASYNC. You can use the commands listed in this section regardless of the BASIC simulation language that you are using.

There are also commands that are specific to a simulation language or protocol. These commands are used in addition to the commands in this section, when you are using the appropriate language and protocol. Refer to the protocol-specific sections for more information.

Index

In this section, the commands are listed in a series of tables by function. This enables you to locate the BASIC command that performs a particular function that you need. For example, if you need for a command that converts a string from ASCII to EBCDIC, you could scan the STRING COMMANDS table and quickly locate the \texttt{EBCS} command.

The tables list the commands, a brief description, and type of command (\texttt{Statement}, \texttt{Direct}, \texttt{Variable}). For more information about a command, refer to the page number indicated.

Program Development

Note that the program development commands on page 3.4-5 cannot be used within a program. They are valid only interactively at the simulation ! prompt.

The functional tables are:

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miscellaneous Commands</td>
<td>3.4-2</td>
</tr>
<tr>
<td>Data File Commands</td>
<td>3.4-2</td>
</tr>
<tr>
<td>Display and Print Commands</td>
<td>3.4-3</td>
</tr>
<tr>
<td>Function Key Commands</td>
<td>3.4-4</td>
</tr>
<tr>
<td>Loops, Subroutines, Conditional Commands</td>
<td>3.4-4</td>
</tr>
<tr>
<td>Mnemonic Commands</td>
<td>3.4-5</td>
</tr>
<tr>
<td>Read-Only Variables</td>
<td>3.4-5</td>
</tr>
<tr>
<td>Program Development Commands</td>
<td>3.4-6</td>
</tr>
<tr>
<td>String Commands</td>
<td>3.4-7</td>
</tr>
<tr>
<td>Video Display Commands</td>
<td>3.4-8</td>
</tr>
<tr>
<td>Trace Buffer Commands</td>
<td>3.4-9</td>
</tr>
</tbody>
</table>
### TABLE 3.4-1: BASIC MISCELLANEOUS COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>3.5-2</td>
<td>Variable</td>
<td>Accesses the array.</td>
</tr>
<tr>
<td>ABS</td>
<td>3.5-3</td>
<td>S</td>
<td>Returns the absolute value of a variable.</td>
</tr>
<tr>
<td>COUPLER</td>
<td>3.5-20</td>
<td>S or D</td>
<td>Initializes the Chameleon for transmission and reception using a specified parameter file.</td>
</tr>
<tr>
<td>FLUSH</td>
<td>3.5-38</td>
<td>S or D</td>
<td>Clears the acquisition buffer.</td>
</tr>
<tr>
<td>LET</td>
<td>3.5-59</td>
<td>S or D</td>
<td>Assigns values to numeric or string variables.</td>
</tr>
<tr>
<td>MENU</td>
<td>3.5-67</td>
<td>D</td>
<td>Exits simulation and displays the Chameleon main menu.</td>
</tr>
<tr>
<td>REC</td>
<td>3.5-81</td>
<td>S</td>
<td>Attempts to receive a frame.</td>
</tr>
<tr>
<td>REM</td>
<td>3.5-82</td>
<td>S</td>
<td>Programmer's remark in a program file.</td>
</tr>
<tr>
<td>RND</td>
<td>3.5-90</td>
<td>S or D</td>
<td>Returns a random number within a specified range.</td>
</tr>
<tr>
<td>SET</td>
<td>3.5-93</td>
<td>S or D</td>
<td>Sets a specified physical interface signal to 1 or 0.</td>
</tr>
<tr>
<td>SETUP</td>
<td>3.5-94</td>
<td>D</td>
<td>Exits simulation prompt (!) and displays the parameter set-up menu.</td>
</tr>
<tr>
<td>STOP</td>
<td>3.5-96</td>
<td>S</td>
<td>Unconditionally stops program execution.</td>
</tr>
<tr>
<td>TEST</td>
<td>3.5-97</td>
<td>S or D</td>
<td>Tests a specified physical interface signal for 1 or 0.</td>
</tr>
</tbody>
</table>

### TABLE 3.4-2: BASIC DATA FILE COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE</td>
<td>3.5-18</td>
<td>S or D</td>
<td>Closes a data file that was opened for input or output.</td>
</tr>
<tr>
<td>EOF</td>
<td>3.5-29</td>
<td>Read-only</td>
<td>Indicates if end-of-file has been reached in a data file.</td>
</tr>
<tr>
<td>OPEN</td>
<td>3.5-76</td>
<td>S or D</td>
<td>Opens a new or existing data file for input or output.</td>
</tr>
<tr>
<td>READ</td>
<td>3.5-80</td>
<td>S</td>
<td>Reads the next record from an input file.</td>
</tr>
<tr>
<td>WRITE</td>
<td>3.5-108</td>
<td>S</td>
<td>Writes a string to an output file.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>PAGE</td>
<td>TYPE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DISP F</td>
<td>3.5-25</td>
<td>S or D</td>
<td>Displays last frame transmitted or received. (Not available in SIMP/L. See RDISPF and TDISPF in SIMP/L chapter.)</td>
</tr>
<tr>
<td>FILES</td>
<td>3.5-35</td>
<td>D</td>
<td>Displays a list of the files on the hard or floppy disk.</td>
</tr>
<tr>
<td>FLIST</td>
<td>3.5-36</td>
<td>S or D</td>
<td>Displays the current function key assignments.</td>
</tr>
<tr>
<td>INPUT</td>
<td>3.5-51</td>
<td>S</td>
<td>Displays a prompt and stores reponse in a variable.</td>
</tr>
<tr>
<td>LFILES</td>
<td>3.5-60</td>
<td>D</td>
<td>Outputs a list of the BASIC files to a printer or remote device.</td>
</tr>
<tr>
<td>LFLIST</td>
<td>3.5-61</td>
<td>S or D</td>
<td>Outputs the current function key assignments to a printer or remote device.</td>
</tr>
<tr>
<td>LIST</td>
<td>3.5-62</td>
<td>D</td>
<td>Displays lines from the program in memory.</td>
</tr>
<tr>
<td>LLIST</td>
<td>3.5-63</td>
<td>D</td>
<td>Outputs lines from the program in memory to a printer or remote device.</td>
</tr>
<tr>
<td>LMLIST</td>
<td>3.5-64</td>
<td>D</td>
<td>Outputs the mnemonic table in memory to a printer or remote device.</td>
</tr>
<tr>
<td>LTPRINT</td>
<td>3.5-66</td>
<td>S or D</td>
<td>Outputs the contents of the trace buffer to a printer or remote device.</td>
</tr>
<tr>
<td>MLIST</td>
<td>3.5-70</td>
<td>S or D</td>
<td>Displays the mnemonic table in memory.</td>
</tr>
<tr>
<td>PRINT</td>
<td>3.5-78</td>
<td>S or D</td>
<td>Displays a string, expression, value, or text.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>3.5-102</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
### TABLE 3.4-4: BASIC FUNCTION KEY COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDEFINE</td>
<td>3.5-34</td>
<td>S or D</td>
<td>Assigns one or more commands to a function key.</td>
</tr>
<tr>
<td>FLIST</td>
<td>3.5-36</td>
<td>S or D</td>
<td>Displays the current function key assignments.</td>
</tr>
<tr>
<td>FLOAD</td>
<td>3.5-37</td>
<td>S or D</td>
<td>Loads a function key definition file into memory.</td>
</tr>
<tr>
<td>FSAVE</td>
<td>3.5-41</td>
<td>S or D</td>
<td>Saves the current function key assignments to disk.</td>
</tr>
<tr>
<td>LFLIST</td>
<td>3.5-61</td>
<td>S or D</td>
<td>Outputs current function key assignments to printer or remote device.</td>
</tr>
</tbody>
</table>

### TABLE 3.4-5: BASIC LOOP, SUBROUTINE, AND CONDITIONAL COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>3.5-12</td>
<td>S</td>
<td>Calls a program from the disk to be executed as a subroutine of the current program being executed. See also EXIT.</td>
</tr>
<tr>
<td>EXIT</td>
<td>3.5-33</td>
<td>S</td>
<td>Returns control from a CALLeD program to the calling program.</td>
</tr>
<tr>
<td>FOR</td>
<td>3.5-39</td>
<td>S</td>
<td>Initiates a loop and determines how many times the loop is repeated. See NEXT.</td>
</tr>
<tr>
<td>GOSUB</td>
<td>3.5-42</td>
<td>S</td>
<td>Sends program control to a specific line number in the program to execute a subroutine. See RETURN.</td>
</tr>
<tr>
<td>GOTO</td>
<td>3.5-43</td>
<td>S</td>
<td>Sends program control to a specific line number in the program.</td>
</tr>
<tr>
<td>IF</td>
<td>3.5-48</td>
<td>S</td>
<td>Allows program flow to be changed based on a decision (conditional branching).</td>
</tr>
<tr>
<td>NEXT</td>
<td>3.5-74</td>
<td>S</td>
<td>Signals the end of a loop and increments the loop counter. See FOR.</td>
</tr>
<tr>
<td>RETURN</td>
<td>3.5-85</td>
<td>S</td>
<td>Returns program control from a subroutine to the statement following the GOSUB command. See GOSUB.</td>
</tr>
</tbody>
</table>
### TABLE 3.4-6: BASIC MNEMONIC COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE</td>
<td>3.5-22</td>
<td>S</td>
<td>Defines a new mnemonic or redefines an existing mnemonic for the mnemonic table.</td>
</tr>
<tr>
<td>DELETE</td>
<td>3.5-24</td>
<td>S or D</td>
<td>Deletes a mnemonic from the mnemonic table.</td>
</tr>
<tr>
<td>FREE</td>
<td>3.5-40</td>
<td>S or D</td>
<td>Read-only variable that returns the number of free mnemonic table entries.</td>
</tr>
<tr>
<td>LMLIST</td>
<td>3.5-64</td>
<td>D</td>
<td>Outputs the mnemonic table in memory to a printer or remote device.</td>
</tr>
<tr>
<td>MLIST</td>
<td>3.5-70</td>
<td>S or D</td>
<td>Displays the mnemonic table in memory.</td>
</tr>
<tr>
<td>MLOAD</td>
<td>3.5-71</td>
<td>S or D</td>
<td>Loads a mnemonic file from disk into memory.</td>
</tr>
<tr>
<td>MSAVE</td>
<td>3.5-72</td>
<td>D</td>
<td>Saves a mnemonic table to disk.</td>
</tr>
</tbody>
</table>

### TABLE: 3.4-7: BASIC READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF</td>
<td>3.5-29</td>
<td>Variable</td>
<td>Indicates end-of-file in an input file.</td>
</tr>
<tr>
<td>FREE</td>
<td>3.5-40</td>
<td>Variable</td>
<td>Returns the number of free mnemonic table entries.</td>
</tr>
<tr>
<td>SIZE</td>
<td>3.5-95</td>
<td>Variable</td>
<td>Returns the size of free program area in bytes.</td>
</tr>
<tr>
<td>TFREE</td>
<td>3.5-98</td>
<td>Variable</td>
<td>Returns length of the unused trace buffer in bytes.</td>
</tr>
<tr>
<td>TIME</td>
<td>3.5-99</td>
<td>Variable</td>
<td>Returns one byte of real time, in specified unit.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>PAGE</td>
<td>TYPE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>AUTO</td>
<td>3.5-6</td>
<td>D</td>
<td>Provides automatic line numbering.</td>
</tr>
<tr>
<td>CHAIN</td>
<td>3.5-15</td>
<td>D</td>
<td>Loads and executes a program from disk.</td>
</tr>
<tr>
<td>EDIT</td>
<td>3.5-27</td>
<td>D</td>
<td>Edits a line from the program in memory.</td>
</tr>
<tr>
<td>ERASE</td>
<td>3.5-32</td>
<td>D</td>
<td>Deletes lines from the program in memory.</td>
</tr>
<tr>
<td>FILES</td>
<td>3.5-35</td>
<td>D</td>
<td>Displays a list of the files on the hard or floppy disk.</td>
</tr>
<tr>
<td>KILL</td>
<td>3.5-56</td>
<td>D</td>
<td>Deletes a file from disk.</td>
</tr>
<tr>
<td>LFILES</td>
<td>3.5-60</td>
<td>D</td>
<td>Outputs file directory to a printer or remote device.</td>
</tr>
<tr>
<td>LIST</td>
<td>3.5-62</td>
<td>D</td>
<td>Displays lines from the program in memory.</td>
</tr>
<tr>
<td>LLIST</td>
<td>3.5-63</td>
<td>D</td>
<td>Outputs lines from the program in memory to a printer or remote device.</td>
</tr>
<tr>
<td>LOAD</td>
<td>3.5-65</td>
<td>D</td>
<td>Loads a program from disk into memory.</td>
</tr>
<tr>
<td>MERGE</td>
<td>3.5-68</td>
<td>D</td>
<td>Combines the program lines in memory with a program file on disk.</td>
</tr>
<tr>
<td>NEW</td>
<td>3.5-73</td>
<td>D</td>
<td>Deletes the program in memory so that a new program can be entered.</td>
</tr>
<tr>
<td>RESEQ</td>
<td>3.5-83</td>
<td>D</td>
<td>Re-numbers line numbers of program in memory to increments of ten.</td>
</tr>
<tr>
<td>RUN</td>
<td>3.5-91</td>
<td>D</td>
<td>Executes the program in memory.</td>
</tr>
<tr>
<td>SAVE</td>
<td>3.5-92</td>
<td>D</td>
<td>Saves the program in memory to disk.</td>
</tr>
<tr>
<td>TROFF</td>
<td>3.5-103</td>
<td>D</td>
<td>Turns program debugging mode off.</td>
</tr>
<tr>
<td>TRON</td>
<td>3.5-104</td>
<td>D</td>
<td>Turns program debugging mode on.</td>
</tr>
</tbody>
</table>
**TABLE 3.4-9: BASIC STRING COMMANDS**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC$</td>
<td>3.5-4</td>
<td>S</td>
<td>Converts a string from EBCDIC to ASCII.</td>
</tr>
<tr>
<td>ATIME$</td>
<td>3.5-5</td>
<td>S</td>
<td>Returns the real time in ASCII.</td>
</tr>
<tr>
<td>BCD$</td>
<td>3.5-7</td>
<td>S</td>
<td>Converts a string variable from ASCII numeric values to BCD.</td>
</tr>
<tr>
<td>CHR$</td>
<td>3.5-16</td>
<td>S</td>
<td>Converts an ASCII value to binary.</td>
</tr>
<tr>
<td>DEC$</td>
<td>3.5-21</td>
<td>S</td>
<td>Creates an ASCII numeric value string of a sufficient length to express the decimal value of exp.</td>
</tr>
<tr>
<td>EBC$</td>
<td>3.5-26</td>
<td>S</td>
<td>Converts a string from ASCII to EBCDIC.</td>
</tr>
<tr>
<td>HEX$</td>
<td>3.5-44</td>
<td>S</td>
<td>Creates an ASCII four-character string which is the hex representation of any valid expression.</td>
</tr>
<tr>
<td>HEX &gt;</td>
<td>3.5-45</td>
<td>S</td>
<td>Assigns the hex representation of an expression to a string.</td>
</tr>
<tr>
<td>INKEY$</td>
<td>3.5-50</td>
<td>S</td>
<td>Assigns the value of the last character typed to a specific string.</td>
</tr>
<tr>
<td>$INPUT</td>
<td>3.5-53</td>
<td>S</td>
<td>Allows the user to enter a string from the keyboard.</td>
</tr>
<tr>
<td>INSTR</td>
<td>3.5-55</td>
<td>S</td>
<td>Returns a numeric value which indicates the offset position of substring str2 within the string str1.</td>
</tr>
<tr>
<td>LEFT$</td>
<td>3.5-57</td>
<td>S</td>
<td>Return the leftmost exp number of characters of the string.</td>
</tr>
<tr>
<td>LEN</td>
<td>3.5-58</td>
<td>S</td>
<td>Returns the length of the string.</td>
</tr>
<tr>
<td>MID$</td>
<td>3.5-69</td>
<td>S</td>
<td>Returns characters from within a string from the position specified by exp1 for the number of characters specified by x.</td>
</tr>
<tr>
<td>RIGHT$</td>
<td>3.5-89</td>
<td>S</td>
<td>Returns the rightmost exp number of characters of the string.</td>
</tr>
<tr>
<td>TIME$</td>
<td>3.5-100</td>
<td>S</td>
<td>Produces a five-byte real-time value: hours, minutes, seconds, hundredths of seconds and tenths of milliseconds.</td>
</tr>
<tr>
<td>VAL</td>
<td>3.5-107</td>
<td>S</td>
<td>Converts the first two characters of a string into their numeric form.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>PAGE</td>
<td>TYPE</td>
<td>ACTION</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>BLK</td>
<td>3.5-8</td>
<td>S or D</td>
<td>Displays text blinking</td>
</tr>
<tr>
<td>BLKHLF</td>
<td>3.5-9</td>
<td>S or D</td>
<td>Displays text blinking and double intensity</td>
</tr>
<tr>
<td>BLKREV</td>
<td>3.5-10</td>
<td>S or D</td>
<td>Displays text blinking and reverse video</td>
</tr>
<tr>
<td>BLKUND</td>
<td>3.5-11</td>
<td>S or D</td>
<td>Displays text blinking, and underlined</td>
</tr>
<tr>
<td>CLS</td>
<td>3.5-19</td>
<td>S or D</td>
<td>Clears screen</td>
</tr>
<tr>
<td>DEL</td>
<td>3.5-23</td>
<td>S</td>
<td>Deletes line</td>
</tr>
<tr>
<td>ERAEOL</td>
<td>3.5-30</td>
<td>S</td>
<td>Erases to end of line</td>
</tr>
<tr>
<td>ERAEOS</td>
<td>3.5-31</td>
<td>S</td>
<td>Erases to end of screen</td>
</tr>
<tr>
<td>HLF</td>
<td>3.5-46</td>
<td>S or D</td>
<td>Displays text in double intensity</td>
</tr>
<tr>
<td>HLFUND</td>
<td>3.5-47</td>
<td>S or D</td>
<td>Displays text underlined in double intensity</td>
</tr>
<tr>
<td>INS</td>
<td>3.5-54</td>
<td>S</td>
<td>Inserts line</td>
</tr>
<tr>
<td>NRM</td>
<td>3.5-75</td>
<td>S or D</td>
<td>Cancels display effects and returns to normal display.</td>
</tr>
<tr>
<td>REV</td>
<td>3.5-86</td>
<td>S or D</td>
<td>Displays text in reverse video</td>
</tr>
<tr>
<td>REVHLF</td>
<td>3.5-87</td>
<td>S or D</td>
<td>Displays text in reverse video and double intensity</td>
</tr>
<tr>
<td>REVUND</td>
<td>3.5-88</td>
<td>S or D</td>
<td>Displays text underlined in reverse video</td>
</tr>
<tr>
<td>UND</td>
<td>3.5-106</td>
<td>S or D</td>
<td>Displays text underlined</td>
</tr>
<tr>
<td>XYPLOT(y,x)</td>
<td>3.5-109</td>
<td>S or D</td>
<td>X,Y screen addressing</td>
</tr>
</tbody>
</table>
### TABLE 3.4-11: BASIC TRACE BUFFER COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>3.5-17</td>
<td>S or D</td>
<td>Clears the trace buffer.</td>
</tr>
<tr>
<td>LTPRINT</td>
<td>3.5-66</td>
<td>S or D</td>
<td>Outputs the contents of the trace buffer to a printer or remote device.</td>
</tr>
<tr>
<td>TFREE</td>
<td>3.5-98</td>
<td>Variable</td>
<td>Read-only variable. Returns the length of the unused trace buffer in bytes.</td>
</tr>
<tr>
<td>TLOAD</td>
<td>3.5-101</td>
<td>S or D</td>
<td>Loads a trace file from disk.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>3.5-102</td>
<td>S or D</td>
<td>Displays contents of the trace buffer.</td>
</tr>
<tr>
<td>TSAVE</td>
<td>3.5-105</td>
<td>S or D</td>
<td>Saves a trace file to disk.</td>
</tr>
</tbody>
</table>

### TABLE 3.4-12: BASIC CONTROL and EDIT KEY COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL C</td>
<td>Aborts program execution that is pausing for input.</td>
</tr>
<tr>
<td>CTRL Q</td>
<td>Restores line in history buffer to the ! input line and in EDIT mode.</td>
</tr>
<tr>
<td>CTRL S</td>
<td>Pauses execution of program. To re-start, press any key.</td>
</tr>
<tr>
<td>CTRL Z</td>
<td>Same as CTRL C</td>
</tr>
<tr>
<td>ESC</td>
<td>Aborts execution of program; returns to ! prompt.</td>
</tr>
</tbody>
</table>

**EDITING CONTROL KEYS**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL D</td>
<td>Deletes character under cursor</td>
</tr>
<tr>
<td>CRTL I</td>
<td>Inserts one character space at cursor position.</td>
</tr>
<tr>
<td>CTRL L</td>
<td>Moves cursor one character to the right.</td>
</tr>
<tr>
<td>CTRL P</td>
<td>Moves cursor to end of line.</td>
</tr>
<tr>
<td>CTRL R</td>
<td>Moves cursor one character to the left.</td>
</tr>
<tr>
<td>CTRL X</td>
<td>Erases complete line, including line number.</td>
</tr>
<tr>
<td>CTRL Z</td>
<td>Exits editing and restores original text.</td>
</tr>
<tr>
<td>←/→</td>
<td>Moves cursor to left or right, respectively.</td>
</tr>
</tbody>
</table>
3.5 BASIC COMMANDS

Introduction

This section provides a complete description of the BASIC commands. The commands are organized alphabetically, one command per page.

If you know the name of the command you want to use, you can look up the command in this section. If you are unsure of the command you need, refer to the previous section which lists the commands by function.

The following information is provided in this section:

- Command description
- Type of command (Statement or Direct)
- Syntax
- Command abbreviation
- Related commands (See Also)
- Example(s) of command usage
@ (ARRAY)

Description

There is one array available for the user, which is referenced by the @ symbol. The maximum number of values allowed in an array is 256, with subscripts starting at zero. Array elements can be used anywhere that normal numeric variables (A-Z) are valid.

Type

Variable

Syntax

@ (exp)

where: @ is the symbol for the array, and exp is any valid expression for the subscript in the array.

Abbreviation

None

See Also

Numeric Variables (See page 3.2-8)

Example 1

This program prints the 256 array elements in descending order.

```
10 FOR A=0 TO 255
20 @(A)=255-A
30 PRINT "Array(",A,")=",@(A)
40 NEXT A
```

!RUN <RETURN>

(result) Array( 0)= 255
        Array( 1)= 254
        Array( 2)= 253
        Array( 3)= 252
        etc.

Example 2

To access a specific element in the array, use the subscript to assign the element of the array to a numeric variable.

```
B = @(8)
```
## ABS

### Description
The ABS command returns the absolute value of a variable. This means that it returns the magnitude of the variable without the sign.

### Type
Statement

### Syntax
`ABS(x)`
where: \( x \) is a number or numeric variable, integer only.

### Abbreviation
AB()

### See Also
None

### Example 1
To display the absolute value of -10, enter:

```
PRINT ABS(-10)
```

(result) 10

### Example 2
This program displays the absolute value of a numeric variable.

```
10 X = 5
20 PRINT ABS(X)
!RUN  <RETURN>
```

(result) 5

### Example 3
This program displays a numeric variable and its absolute value.

```
10 X = -5
20 Y = ABS(X)
20 PRINT X, Y
!RUN  <RETURN>
```

(result) -5 5
ASC$ is a string function that returns the ASCII code character representation of an EBCDIC string.

Statement

ASC$(x)$

where: $x$ is an EBCDIC string.

None

EBC$

This example displays an EBCDIC value and its ASCII value in hex. and string variable

10 $A=CHR$(&FO) String variable $A$ is assigned an EBCDIC value.
20 $B=ASC$(A) $B$ is assigned the ASCII value of $A$.
30 PRINT %$A," ",%$B Both values are printed in hex.
!RUN <RETURN>

(result) 41 42 43 44 45 C1 C2 C3 C4 C5 41 42 43 44 45

Example 2

String variable $A$ is assigned an ASCII value.

20 $B = EBC$(A) $B$ is assigned the EBCDIC value of $A$.
30 $C = ASC$(A) $C$ is assigned the ASCII value of $B$.
40 PRINT %$A, %$B, %$C All three values are printed in hex.

!RUN <RETURN>

(result) 41 42 43 44 45 C1 C2 C3 C4 C5 41 42 43 44 45
ATIME$ is a string function that returns the ASCII value of the realtime stamp, converted from its Binary Coded Decimal format. The time is given in units of hours, minutes, seconds, .1 seconds, .01 seconds, .001 seconds, and .0001 seconds.

Note: The clock interval in simulation is 10 milliseconds. However, when the time is read, it is accurate to one tenth of a millisecond (100 microseconds).

See Volume I, Section x.x for an explanation of time stamping in Analysis mode, in which accuracy is to 20 microseconds.

Type: Statement

Syntax: ATIME$

Abbreviation: None

See Also: TIME, TIME$

Example: This example displays the current time.

10 $B = ATIME$
20 PRINT $B
!RUN <RETURN>

(result) 1220305060

The time is 12:20:30:50:60

- hours
- minutes
- seconds
- .1 seconds
- .01 seconds
- .001 seconds
- .0001 seconds
- .001 seconds
- .0001 seconds
- .0001 seconds
- .0001 seconds
AUTO

Description
The AUTO command provides automatic line numbering while you are entering program code.

Type
Direct

Syntax
AUTO       Starts at line 10 and increments by 10.
AUTO x     Starts at line x and increments by 10.
AUTO x,y   Starts at line number x and increments by y.

Use caution when you have program lines in memory. Any new lines using the same line numbers as existing lines will overwrite the original lines.

To terminate automatic numbering, press the RETURN key or CTRL Z when a line number only is displayed.

Abbreviation
A.

Example 1
To have automatic line numbering beginning with 10 in increments of 10, enter:

AUTO

(result)  10   ...
         20   ...
         30   ...
     etc.

Example 2
To have automatic line numbering beginning with 100 in increments of 5, enter:

AUTO 100.5

(result) 100   ...
        105   ...
        110   ...
    etc.
BCD$

Description
BCD$ is a string function that converts an ASCII string to Binary Coded Decimal (BCD).
This function is not available in FRAMEM.

Type
Statement

Syntax
BCD$(x)
where: $x$ is an ASCII string.

Abbreviation
None

See Also
ASC$

Example
This example displays the BCD value of an ASCII string.

```
10 $A="12345"    Assigns ASCII string to $A.
20 $B=BCD$(A)    Assigns the BCD value of $A$ to $A$.
30 PRINT %$B    Displays $B$ in hex.
!RUN <RETURN>
```

(result)  12 34 50

Note that a zero is added to the end of the string to complete the byte.
BLK

Description  The BLK command causes text to blink on the Chameleon screen. Use the NRM command to cancel this and all other special display effects and return to normal display mode.

Type  Direct or Statement

Syntax  BLK

Abbreviation  None

See Also  NRM, BLKHLF, BLKREV, BLKUND

Example 1  This example causes the simulation prompt and subsequent characters to blink.

```
10 BLK
```

(result)  OK!  (appears blinking on the screen)

Example 2  This example changes to blinking display and then resets the screen to normal display.

```
10 BLK
20 PRINT "I'M BLINKING!"
30 NRM
!RUN  <RETURN>
```

(result)  I'M BLINKING  (appears blinking on the screen)
BLKHLF

Description: The BLKHLF command causes text to blink in double intensity on the Chameleon screen. Use the NRM command to cancel blinking and all other special display effects.

Type: Direct or Statement

Syntax: BLKHLF

Abbreviation: BLKH.

See Also: NRM, BLK, BLKREV, BLKUND

Example 1: This example causes the simulation prompt to blink in double intensity.

```
BLKHLF
(RESULT) OK! (appears blinking in double intensity)
```

Example 2: This example causes text to blink in double intensity and then resets the screen to normal display.

```
10 BLKHLF
20 PRINT "I'M BLINKING!"
30 NRM
!RUN <RETURN>
```

(result) I'M BLINKING (appears blinking in double intensity on the screen)
# BLKREV

**Description**
The BLKREV command causes text to blink in reverse video on the Chameleon screen. Use the NRM command to cancel this and all other special display effects.

**Type**
Direct or Statement

**Syntax**
BLKREV

**Abbreviation**
BLKR.

**See Also**
NRM, BLK, BLKHLF, BLKUND

**Example 1**
This example causes the simulation prompt to blink in reverse video.

```basic
BLKREV
```

(result)  OK!  (appears blinking in reverse video)

**Example 2**
This example causes text to blink in reverse video and then resets the screen to normal display.

```basic
10  BLKREV
20  PRINT "I’M BLINKING!"
30   NRM
!RUN  <RETURN>
```

(result)  I’M BLINKING  (appears blinking in reverse video)
BLKUND

Description
The BLKUND command causes text to blink and be underlined on the Chameleon screen. Use the NRM command to cancel this and all other special display effects.

Type
Direct or Statement

Syntax
BLKUND

Abbreviation
BLKU.

See Also
NRM, BLK, BLKHLF, BLKREV

Example 1
This example causes the simulation prompt to blink and be underlined.

    BLKUND
    OK!  (appears blinking and underlined)

Example 2
This example causes text to blink and be underlined and then resets the screen to normal display.

    10  BLKUND
    20  PRINT "I'M BLINKING!"
    30  NRM
            !RUN   <RETURN>

    (result)  I'M BLINKING  (appears blinking and underlined)
CALL

Description
The CALL command calls a program from the disk as if it were a subroutine. It causes a separate program to be executed as part of the main program.

When a program is called, all variables, string values, and transmission variables retain the values they had in the main program. If the called program alters any variables, they retain these new values when control is returned to the main program. In other words, all of the variables are global.

A called program can call other programs. Chameleon BASIC supports up to 16 levels of calling. The main program is saved automatically in a temporary file. There must be sufficient room on your hard or floppy disk for the main program to be saved.

The EXIT command is used in a called program to return control to the main program.

When calling a program, open data files are automatically closed. To reopen them to append data, use the OPEN command.

Type
Statement

Syntax
CALL "name"
CALL "A:name"
CALL "B:name"
$A = "name"
CALL $A

where: name is a valid program filename.

Abbreviation
CA.

See Also
EXIT, OPEN
Example 1

This example demonstrates the use of CALL and EXIT. PROG1 is the main program and calls PROG2, as shown below:

```
10 PRINT "THIS IS THE MAIN PROGRAM"
20 CALL "PROG2"
30 PRINT "BACK AGAIN"
```

PROG2 is called by PROG1, prints a message and returns control to PROG1, as shown below:

```
10 PRINT "THIS IS THE CALLED PROGRAM"
20 EXIT
```

(result) When PROG1 is run, the following messages are displayed:

```
THIS IS THE MAIN PROGRAM
THIS IS THE CALLED PROGRAM
BACK AGAIN
```

Example 2

This example demonstrates how variables retain their values when a program is called or exited. PROG1 prints the value of variable A and calls PROG2, as shown below:

```
10 A = 5
20 PRINT A
30 CALL "PROG2"
40 PRINT A
```

PROG2 prints the value of variable A, which demonstrates that the value of the variable is retained and passed to the called program. The called program then increments variable A and returns control to PROG1. PROG2 is shown below.

```
10 PRINT A
20 A = A + 1
30 EXIT
```

When PROG1 is run, the following results are displayed:

```
5 (Output from the main program)
5 (Output from the called program)
6 (Output from the main program)
```
Example 3

This example demonstrates different levels of calling. The main program, shown below, calls 15 different programs.

```basic
10  REM ******** THIS IS THE MAIN PROGRAM ********
20  A=0
40  XYPLOT(0,0)
50  CLS
60  $Z = "THIS PROGRAM SHOWS MULTI-LEVEL CALLING"
70  B=LEN($Z)/2
80  XYPLOT(0,B)
90  PRINT $Z,
100 XYPLOT(2,0)
120 PRINT "HERE WE GO"
130 CALL "PROGA"
140 XYPLOT(2,40)
150 PRINT "WE'RE FINISHED"
160 XYPLOT(19,0)
```

The 15 programs that are called are identical, except for their filenames, which are PROGA through PROGO. The code for the called programs is shown below.

```basic
10  XYPLOT(5+A,1)
20  PRINT "THIS IS SUB-PROGRAM NUMBER",A
30  IF A = 14 EXIT
40  A = A + 1
50  $B = "PROG" + CHR$(A+&41)
60  CALL $B
70  A = A - 1
80  XYPLOT(5+A,40)
90  PRINT "NOW BACK IN PROG", A
100 EXIT
```

When the main program is run, the following series of messages is displayed on the screen. The right side lines are displayed first one by one down the screen, then the left side lines are displayed one by one up the screen.

```
THIS PROGRAM SHOWS MULTI-LEVEL CALLING

HERE WE GO

WE'RE FINISHED

THIS IS SUB-PROGRAM NUMBER 0  NOW BACK IN PROG 0
THIS IS SUB-PROGRAM NUMBER 1  NOW BACK IN PROG 1
THIS IS SUB-PROGRAM NUMBER 2  NOW BACK IN PROG 2
THIS IS SUB-PROGRAM NUMBER 3  NOW BACK IN PROG 3
... ...
... ...
THIS IS SUB-PROGRAM NUMBER 12  NOW BACK IN PROG 12
THIS IS SUB-PROGRAM NUMBER 13  NOW BACK IN PROG 13
THIS IS SUB-PROGRAM NUMBER 14
```

Tekelec 3.5-14  6/11/90
CHAIN

Description
The CHAIN command loads a specified program file from the disk into memory and runs it. This is the same as using a LOAD command and then a RUN command.

Type
Statement or Direct

Syntax
CHAIN"name"
Loads and runs program from the default drive (A).
CHAIN "A:name"
Loads and runs program from the hard disk drive (A).
CHAIN "B:name"
Loads and runs program from the floppy disk drive (B).
$A = "name"
CHAIN $A
Loads and runs program name stored in $A from the default drive (A).

where: name is a the program filename to load and run.

Abbreviation
CH.

See Also
LOAD, RUN

Examples
These examples show two different ways to load and run a file called PROG1 from the floppy disk drive.

CHAIN"B:PROG1"

10 $A = "B:PROG1"
20 CHAIN $A
CHRS$  

**Description**  
CHRS$ is a string function that stores the binary equivalent of an ASCII value. When the string variable is printed to the screen, the ASCII character is displayed.

**Type**  
Statement

**Syntax**  
CHRS$(exp)

where: exp is a number, numeric variable, arithmetic or logical function.

**Abbreviation**  
None

**Example 1**  
This example stores and prints the binary equivalent of the ASCII character 65 as variable $C$.

```
10 $C = CHR$(65)
20 PRINT $C
!RUN <RETURN>
```

(result) A  (ASCII equivalent of binary 65)

**Example 2**  
This example stores hex values to numeric strings. Logical OR is used to define the value of string variable $C$.

```
10 A=&40
20 B=&01
30 $C = CHR$(A OR B)
40 PRINT $C
!RUN <RETURN>
```

(result) A

**Example 3**  
This example uses a FOR loop to store the entire ASCII table from 11 hex to 7E hex in string variable $A$.

```
5 $A = ""
10 FOR X = 17 TO 126
20 $A = $A + CHR$(X)
30 PRINT $A
40 NEXT X
50 STOP
!RUN <RETURN>
```

The values are stored in binary code, but are displayed in ASCII because PRINT $A$ converts binary values to ASCII.

""#$%&'()*+,-./0123456789:;<=?>@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_abcdefghijklmnopqrstuvwxyz{|}~"
CLEAR

Description
The CLEAR command clears the trace buffer. The trace buffer contains the data after a REC command is executed. See Section 3.2 for more information about simulation buffers.

Type
Statement or Direct

Syntax
CLEAR

Abbreviation
CL.

See Also
TSAVE, TLOAD, TPRINT

Example
This example clears the trace buffer and then displays it.

10 CLEAR
20 TPRINT
IRUN <RETURN>

(result)  !

Nothing is displayed, since the trace buffer was cleared of all data and is now empty.
CLOSE

Description: The CLOSE command closes data files that were opened for input or output. A file opened for input does not need to be closed before returning to the menu system or before powering off the Chameleon. Use of the optional argument I or O will close only that data file opened for input (I) or output (O). With no argument, all open data files will be closed.

Type: Statement or Direct

Syntax:
- CLOSE Closes all open data files
- CLOSE I Closes all data input files
- CLOSE O Closes all data output files

Abbreviation: CLO.

See Also: OPEN, READ, WRITE, EOF

Example:
- To close all open data files, enter:
  CLOSE

- To close the data input file, enter:
  CLOSE I

- To close the data output file, enter:
  CLOSE O
CLS

Description
The CLS command clears the Chameleon screen.

Type
Statement or Direct

Syntax
CLS

Abbreviation
None

See Also
INS, DEL, ERAEOL, ERAEOS

Example
CLS
(result)

The Chameleon screen is cleared of all text and the ! prompt is displayed at the top of the screen.
COUPLER

Description
The COUPLER command loads a parameter set-up file which has been previously saved to disk. When you first start a simulator, the set-up file named DEFAULT is automatically loaded and the Front End Processor (FEP) is initialized using those values. If the FEP is not initialized, when you try to transmit or receive data, the following message is displayed:

P1 SETUP NOT PERFORMED

If you use the SETUP command to display the parameter set-up menu, the DEFAULT values are loaded into the menu and the FEP is reinitialized with those values. To view the values in a parameter set-up file other than DEFAULT, you must do the following:

1. Type SETUP to display the parameter set-up menu.
2. Press S to access the Save Parameters menu.
3. Press E to load a file into the set-up menu.
4. Enter the disk drive and name of the set-up file.

Notes
CLOSE all data files that are open before executing the COUPLER command, or they may be lost.

The COUPLER command is not available in FRAMEM DMI.

Type
Statement or Direct

Syntax
COUPLER "filename"

$A = "name"
COUPLER $A

where: name is the parameter setup filename.

Abbreviation
CO.

See Also
SETUP, CLOSE

Example 1
This example loads the a parameter set-up file named TEST1 from the hard disk.

COUPLER "TEST1"
DECS

Description

DECS is a string function that converts a valid numeric expression into a string of ASCII decimal characters. Hexadecimal entries return positive ASCII decimal characters in the range of 0 to 7FFF hexadecimal. 8000 to FFFF hexadecimal will return -32768 to -1 respectively.

Type

Statement or Direct

Syntax

$X = DECS(exp)

where: exp is a number, numeric variable, arithmetic function, or logical function.

Abbreviation

None

Example 1

To assign the ASCII digits "10" to string A, enter:

$A = DECS(10)
PRINT $A

(result) 10

Example 2

To assign the ASCII digits "-10" to string A from an arithmetic expression, enter:

$A = DECS(((10 * 2) - 30))
PRINT $A

(result) -10

Example 3

10 $A = DECS(&FF)
20 PRINT $A
!RUN <RETURN>

(result) 255

Example 4

10 A = &10
20 B = &01
30 $A = DECS(A AND B)
40 PRINT $A
!RUN <RETURN>

(result) 0
DEFINE

Description

The DEFINE command defines a new mnemonic or re-defines an existing mnemonic for the mnemonic table. Refer to Section 3.2 for a general discussion of the use of mnemonic tables in simulation.

The structure and function of mnemonics is protocol specific, therefore you should refer to the appropriate chapter for a description of the DEFINE syntax for the simulator you are using.
## DEL

**Description**
The DEL command deletes a line from the Chameleon screen. To erase a line of text without deleting the line, refer to the ERAEOL command.

**Type**
Statement

**Syntax**
DEL

**Abbreviation**
None

**See Also**
ERAEOL, ERAEOS, XYPlot

**Example**
This example displays text on line 20 of the Chameleon screen and then deletes line 0 (top line) of the screen 22 times. As each line is deleted, the message moves toward the top of the screen.

```
10CLS          Clears the screen.
20XYPlot(20,0) Positions cursor on line 20 of screen.
30PRINT "HELLO" Prints message on line 20.
40XYPlot(0,0)  Positions cursor on line 0 of screen.
50FOR A = 1 TO 22 Sets up loop to repeat 22 times.
60DEL          Deletes line 0 from screen.
70NEXT A       Increments loop counter.
!RUN <RETURN>
```
## DELETE

**Description**
The DELETE command removes a mnemonic entry from the mnemonic table.

**Type**
Statement or Direct

**Syntax**
```
DELETE "name"
```
where: *name* is the name of the mnemonic.

**Abbreviation**
D.

**See Also**
DEFINE, MLIST, MSAVE, MLOAD

**Example**
To delete a mnemonic entry named NEW, enter:
```
DELETE "NEW"
```
## DISPF

### Description
The DISPF command displays the last frame transmitted or received. There is a buffer that contains the last frame transmitted or received and the DISPF command displays the contents of that buffer.

After a REC command, the buffer is emptied. If a frame was received, it is put into the buffer; if nothing was received, the buffer will be empty. The length of this buffer varies with the protocol being used.

Refer to section 3.2 for information about simulation buffers.

**Notes:** DISPF is not available in Async.

SIMP/L uses a variation of the DISPF command as described in the syntax below.

### Type
Statement or Direct

### Syntax
- **DISPF** Syntax for all BASIC simulators except SIMP/L and Async.
- **RDISPF** In SIMP/L only, RDISPF displays the last frame/packet/message received.
- **TDISPF** In SIMP/L only, TDISPF displays the last frame/packet/message transmitted.

### Abbreviation
DISP.

### See Also
LDISPF

### Example
This FRAMEM LAPD example transmits two strings and displays the contents of the DISPF buffer and the trace buffer.

```
10 $A = "hello"  Assigns string to $A.
20 TRAN $A       Transmits $A.
30 DISPF         Displays the DISPF buffer ($A).
40 $B = "goodbye" Assigns string to $B.
50 TRAN $B       Transmits $B.
60 DISPF         Displays the DISPF buffer ($B).
70 TPRINT        Displays the trace buffer ($A and $B).
!RUN <RETURN>
```

(result):

```
68656C6C6F
676F66666C6F
```

6/11/90
## EBCS

**Description**
EBCS is a string function that converts an ASCII string to EBCDIC. ASCII/EBCDIC conversion charts are in the Chameleon User's Manual, Volume I, Appendix A.

**Type**
Statement or Direct

**Syntax**
$A = EBCS(B)$

where: $B$ is an ASCII string.

**Abbreviation**
None

**Example**
To assign string A the EBCDIC equivalent of the ASCII characters ABCD, enter:

```
10 $B = "ABCD"
20 $A = EBCS($B)
30 PRINT % $A
```

!RUN <RETURN>

(result) C1 C2 C3 C4
EDIT

Description  The EDIT command allows you to edit a line from the program in memory. When you enter the EDIT command, the program line is placed in a buffer and the line number is displayed on the screen. Use the edit commands listed below to make the changes you need to the program line. You can also change a program line by re-entering the line entirely.

Note that the edit command cannot be used with unnumbered commands entered at the ! prompt. However, there is a history buffer that stores the last command entered at the ! prompt. CTRL Q restores the line from the buffer so that you can edit it and enter is again.

Type  Direct

Syntax  EDIT x

where: x is the line number to edit.

Abbreviation  E.

See Also  SAVE, ERASE, LOAD

Example  EDIT 20

Edit Commands  Use the following command to edit a program line after the EDIT command is executed. The editing commands affect either the line as it is displayed on the screen, or the line as it is stored in memory. The description of each command indicates which is affected.

LEFT Arrow and Delete  These keys move the cursor one space to the left and/or hide the character as it appears on the screen. The character remains stored in memory until the new line is saved or until it is deleted using CTRL + D.

RIGHT Arrow and CTRL + L  The RIGHT arrow key displays the next character of the currently stored line and moves the cursor one character to the right.

CTRL + P  CTRL + P displays the entire line to the right of the cursor, as it is currently stored. The cursor moves to the end of the line.
CTRL + X  
CTRL + X erases the entire line, including line number, from the screen and returns the cursor to the first column. The line remains in memory although it is not displayed on the screen.

CTRL + D  
CTRL + D deletes the next un-displayed character from memory. The deleted character cannot be re-displayed. Entering new code over un-displayed code replaces the old code with the new one in memory.

CTRL + I  
CTRL + I inserts a space to hold an additional character. The cursor moves right one character and a space appears on the display. Use the LEFT arrow key or BACKSPACE key to hide the space, then enter the new character.

RETURN  
The RETURN key saves the line to the left of the cursor, as it appears on the screen. The text right of the cursor is deleted from memory.

CTRL + Z  
CTRL + Z exits from edit mode without saving changes you may have made to the line; only the original code is saved.

Example  
The first column shows the editing command that is entered. The caret symbol (^) indicates a control (CTRL) character.

The second column shows the line as it is displayed on the screen after the command is executed. The tilde symbol (~) indicates the position of the cursor on the display. The third column shows the line in the editing memory.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>DISPLAY</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 EDITING COMMANDS</td>
<td>20 EDITING COMMANDS</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>!EDIT 20</td>
<td>20 ^</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>&quot;P</td>
<td>20 EDITING COMMANDS*</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>Delete</td>
<td>20 EDITING COMMANDS*</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>3 left arrows</td>
<td>20 EDITING COMMANDS*</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>&quot;L</td>
<td>20 EDITING COMMANDS*</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>3 right arrows</td>
<td>20 EDITING COMMANDS*</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>Return key</td>
<td>!</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>!EDIT 20</td>
<td>20 ^</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>&quot;P</td>
<td>20 EDITING COMMANDS*</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>&quot;X</td>
<td>~</td>
<td>20 EDITING COMMANDS</td>
</tr>
<tr>
<td>5 NEW LINE</td>
<td>5 NEW LINE*</td>
<td>5 NEW LINE</td>
</tr>
<tr>
<td>Return key</td>
<td>!</td>
<td>20 EDITING COMMANDS</td>
</tr>
</tbody>
</table>
EOF

Description
EOF is a read-only variable that indicates that the end-of-file condition has been detected in an input (I) file. Opening an input file automatically resets the EOF flag. It has the following values:

- EOF = 0  
  End of data file has not been reached
- EOF = 255  
  End of data file has been reached.

Type
Statement or Direct

Syntax
(See examples below.)

Abbreviation
None

See Also
OPEN, CLOSE

Example 1
PRINT EOF

Example 2
10 A=EOF
20 PRINT A
!RUN  <RETURN>

Example 3
Below are three ways to test for EOF and print a message when the end of file has been reached.

a) IF EOF PRINT "END OF FILE"
b) IF EOF=255 PRINT "END OF FILE"
c) IF EOF # 0 PRINT "END OF FILE"
### ERAEOL

**Description**
The ERAEOL command erases text to the end of the current line.

**Type**
Statement

**Syntax**
ERAEOL

**Abbreviation**
None

**See Also**
ERAEOS, XYPLOT, DEL

**Example**
This example prints a message and then erases the message.

```
10 CLS      ; Clears the screen.
20 XYPLOT(0,0) ; Positions cursor at top of screen.
30 PRINT "HELLO WORLD" ; Prints message at top of screen.
40 XYPLOT(0,6) ; Positions cursor in middle of message.
50 ERAEOL    ; Erases from cursor to end of line.
!RUN <RETURN>
```
## ERAEOS

### Description
The ERAEOS command erases text to the end of the screen.

### Type
Statement

### Syntax
ERAEOL

### Abbreviation
ERAE.

### See Also
ERAEOL, DEL, XYPLOT

### Example
This example prints messages on two lines, and then erases from the second message to the end of the screen.

```plaintext
10 CLS Clears the screen.
20 XYPLOT(5.0) Positions cursor on line 5.
30 PRINT "MESSAGE 1" Prints a message on line 5.
40 XYPLOT(10,0) Positions cursor on line 10.
50 PRINT "MESSAGE 2" Prints a message on line 10.
60 XYPLOT(10,0) Positions cursor at beginning of line 10.
70 ERAEOS Erases from line 10 to the end of the screen.

!RUN <RETURN>
```
ERASE

Description
The ERASE command deletes one or more program lines from the program in memory.

Type
Direct

Syntax
ERASE x,y

where: x is the first line number and y is the last line number you want to erase. All lines between x and y, inclusive, are deleted.

Abbreviation
ER.

See Also
EDIT, LIST, RESEQ

Example 1
To erase all lines between lines 20 and 40, inclusive, enter:
ERASE 20,40

Example 2
To erase line 100 only, enter:
ERASE 100,100

Alternately, you can erase a program line, by entering the line number only and pressing RETURN. For example, to erase line 100, you could enter:
100 <RETURN>
EXIT

Description
The EXIT command returns control to a calling program from a program that has been called using the CALL command.

Type
Statement

Syntax
EXIT

Abbreviation
EX.

See Also
CALL

Example
ProgramA prints a message and then calls ProgramB. ProgramB prints two messages and then executes the EXIT command, returning control to ProgramA to print a final message.

10 REM THIS IS PROGRAMA
20 PRINT "HELLO FROM PROGRAMA"
30 CALL "PROGRAMB"
40 PRINT "GOODBYE FROM PROGRAMA"

10 REM THIS IS PROGRAMB
20 PRINT "HELLO FROM PROGRAMB"
30 PRINT "GOODBYE FROM PROGRAMB"
40 EXIT

(result) HELLO FROM PROGRAMA
HELLO FROM PROGRAMB
GOODBYE FROM PROGRAMB
GOODBYE FROM PROGRAMA
**FDEFINE**

**Description**
The FDEFINE command enables you to define the functions of the 10 keys in the top row of the keyboard (F1 - F10) for special functions.

FDEFINE assigns one or more commands to a function key. Pressing a function key causes the assigned keystrokes to be played back. This simplifies the task of entering repetitive or complex commands or expressions.

**Type**
Statement or Direct

**Syntax**

```
FDEFINE KEYx = "statement statement"
```

where: x is the key number between 1 and 10. Do not insert a blank space between the key number and the equal sign.

The tilde symbol (``) is used to mark the beginning and end of the function key assignment. The caret symbol (``) represents a carriage return and is used between commands. The use of these symbols enables you to include quotation marks and commas as part of the function key assignment.

**Abbreviation**
FD.

**See Also**
FSAVE, FLOAD FLIST

**Example 1**
To have function key F1 list the mnemonic table, the files on the hard disk, and the function key assignments, enter:

```
FDEFINE KEY1="MLIST\FILES\FLIST"
```

**Example 2**
To define function key F2 to load a parameter file called PARA1 and initiate auto line numbering, enter:

```
FDEFINE KEY2="COUPLER="PARA1" AUTO"
```

**Example 3**
To define function key F3 to display GOOD MORNING on the screen, enter:

```
FDEFINE KEY3="PRINT"GOOD MORNING"
```
FILES

Description  The FILES command displays the names of the Chameleon BASIC files on a specified disk drive. The FILES command lists:

- Program Files
- Trace Files
- Mnemonic Table Files
- Data Files
- Setup Files
- Function Key Files

Type  Direct

Syntax  FILES

Lists files on the default drive A (hard disk).
FILES A

Lists files on drive A (hard disk).
FILES B

Lists files on drive B (floppy disk).

Abbreviation  F.

See Also  LFILES, KILL

Example  To list the files on the floppy drive, enter:

FILES B

(result)  
Program Files
: PROGA PROGB PROGC PROGD PROGE
Trace Files
Mnemonic Table Files
Data Files
: DATA1 DATA2
Setup Files
: DEFAULT SETUP1 SETUP2 SETUP3
Function Key Files
**FLIST**

**Description**  The FLIST command lists the ten function keys and their current assignments on the screen.

**Type**  Statement or Direct

**Syntax**  FLIST

**Abbreviation**  FL.

**See Also**  LFLIST, FDEFINE

**Example**  To list the function key assignments in the current function key file, enter:

```
!FLIST
```

(result)

<table>
<thead>
<tr>
<th>KEY</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MLIST*FILES*FLIST*</td>
</tr>
<tr>
<td>2</td>
<td>COUPLER=PARA1*AUTO*</td>
</tr>
<tr>
<td>3</td>
<td>?&quot;GOOD MORNING&quot;*</td>
</tr>
<tr>
<td>4</td>
<td>A+B/2*</td>
</tr>
<tr>
<td>5</td>
<td>LLIST*</td>
</tr>
<tr>
<td>6</td>
<td>LOAD &quot;B:TEST1&quot;*RUN*</td>
</tr>
<tr>
<td>7</td>
<td>FLUSH*NEW*AUTO*</td>
</tr>
<tr>
<td>8</td>
<td>$A=&quot;HELLO&quot;*</td>
</tr>
<tr>
<td>9</td>
<td>TRAN*GOTO10*</td>
</tr>
<tr>
<td>10</td>
<td>RESEQ*LIST*</td>
</tr>
</tbody>
</table>

Note that the tildes that are used to define the keys in the FDEFINE command are not displayed with FLIST.
### FLOAD

**Description**
The FLOAD command loads a function key definition file into memory. The file must already have been saved to disk using FSAVE.

FLOAD redefines all function keys as specified in the loaded function key file, and will redefine keys already FDEFINED.

**Type**
Statement or Direct

**Syntax**
- `FLOAD "name"
- `FLOAD "A:name"
- `FLOAD "B:name"
- `$A = "name"
- `FLOAD $A

*where*: `name` is the function key filename.

**Abbreviation**
FLO.

**See Also**
FSAVE, FDEFINE

**Example**
This example loads the function key file named FKEYS1 from the floppy disk drive into memory.

`FLOAD "B:FKEYS1"`
## FLUSH

**Description**  
The FLUSH command clears the acquisition buffer, the buffer that receives data from the line. For a description of the simulation buffers, refer to section 3.2.

*The FLUSH command is not available in FRAMEM DMI.*

**Type**  
Statement or Direct

**Syntax**  
FLUSH

**Abbreviation**  
FLU.

**See Also**  
CLEAR, REC

**Example**  
FLUSH
### FOR

**Description**
The FOR command controls looping in programs.

**Type**
Statement

**Syntax**

```
FOR x = exp1 TO exp2 [STEP exp3]

NEXT x
```

- **exp1** is a numeric variable and **exp1**, **exp2**, and **exp3** are any valid numeric expressions. **exp1** defines the beginning value of **x**. **exp2** defines the maximum value for **x** to continue executing the loop. **exp3** defines the amount that **x** is incremented each time the loop is executed. **STEP** is optional; the default value is 1.

The NEXT command increments **x** by the **STEP** and marks the end of the loop.

**Abbreviation**
None

**See Also**
NEXT

**Example**

```
10 FOR x = 1 TO 10 STEP 2
20 PRINT x
30 NEXT x
40 PRINT "GOODBYE"
```

(result) 1 3 5 7 9 GOODBYE

This assigns the value of one to **x**. When the program reaches the NEXT x command, it increments **x** by two and sends the program back to the FOR command to execute the loop again. When **x** exceeds 10, the loop terminates and the statement following the NEXT x command is executed.
### FREE

**Description**  
FREE is a read-only variable that returns the number of free mnemonic table entries. A mnemonic table can contain a maximum of 255 entries.

**Type**  
Statement or Direct

**Syntax**  
(See examples below.)

**Abbreviation**  
F.

**See Also**  
MLIST, DELETE, DEFINE, MSAVE, MLOAD

**Example 1**  
To display the number of free entries in the mnemonic table in memory, enter:

```
PRINT FREE
```

**Example 2**  
To print a message based on the number of entries available, you could enter:

```
IF FREE > 0 PRINT "There are ",FREE," entries available."
```

(result)  
There are xxx entries available.

(xxx is the number of free entries)
FSAVE

Description
The FSAVE command saves function key assignments that have been defined with the FDEFINE command to disk.

Type
Statement or Direct

Syntax
FSAVE "name"
FSAVE "A:name"
FSAVE "B:name"
$A = "name"
FSAVE $A

where: name is the function key filename.

Abbreviation
FSA.

See Also
FDEFINE, FLOAD

Example
To saves the current function key assignments in a function key file named FKEYS1 on the floppy disk drive, enter:

FSAVE"B:FKEYS1"
GOSUB

Description
The GOSUB command transfers program control to a specific line number in the program to execute a subroutine. A subroutine ends with a RETURN command, which returns control to the line number following the GOSUB.

Note: Expressions used as line numbers will not be re-evaluated by the RESEQ command.

Type
Statement

Syntax
GOSUB exp

where: exp is the line number, or a valid expression for the line number, of the subroutine.

Abbreviation
GOS.

See Also
RETURN

Example 1
10 FOR A = 1 TO 3
20 GOSUB 200
30 NEXT A
40 STOP
200 PRINT "The value of A = ",A
210 RETURN
!RUN <RETURN>

(result) The value of A = 1
The value of A = 2
The value of A = 3

Example 2
In this example, the subroutine is defined by the expression A*100.

10 FOR A = 1 TO 3
20 GOSUB A*100
30 NEXT A
40 STOP
100 PRINT "SUBROUTINE ON LINE 100"
110 RETURN
200 PRINT "SUBROUTINE ON LINE 200"
210 RETURN
300 PRINT "SUBROUTINE ON LINE 300"
310 RETURN
!RUN <RETURN>

(result) SUBROUTINE ON LINE 100
SUBROUTINE ON LINE 200
SUBROUTINE ON LINE 300
GOTO

Description
The GOTO command transfers program control to a specific line number in the program.

Note: Expressions used as line numbers will not be re-evaluated by the RESEQ command.

Type
Statement

Syntax
GOTO exp

where: exp is the line number, or a valid expression for the line number.

Abbreviation
GO.

See Also
GOSUB

Example 1
10 FOR A = 1 TO 3
20 GOTO 200
30 NEXT A
40 STOP
200 PRINT "The value of A = ",A
210 GOTO 30
!RUN <RETURN>

(result) The value of A= 1
        The value of A= 2
        The value of A= 3

Example 2
In this example, the line is defined by the expression A*100.

10 FOR A = 1 TO 3
20 GOTO A*100
30 NEXT A
40 STOP
100 PRINT "LINE 100"
110 GOTO 30
200 PRINT "LINE 200"
210 GOTO 30
300 PRINT "LINE 300"
310 GOTO 30
!RUN <RETURN>

(result) LINE 100
        LINE 200
        LINE 300
### HEX$

**Description**
HEX$ is a string function that creates an ASCII four-character string which is the hexadecimal representation of any valid expression.

**Type**
Statement

**Syntax**
$A = \text{HEXS}(\text{exp})$

*where: exp is any valid numeric expression.*

**Abbreviation**
None

**See Also**
HEX>

**Example 1**

```
10 $A = \text{HEXS}(255)
20 \text{PRINT }$A
!RUN <RETURN>
```

Result: 00FF

**Example 2**

This program assigns the hexadecimal equivalents of 0 to 255 to string variable $A$ and values 8000 to 80FF to string variable $B$.

```
10 FOR \text{A} = 0 TO 255
20 $A = \text{HEXS}(A)
30 $B = \text{HEXS}(A + &8000)
40 \text{PRINT }$A,", "$B
50 NEXT \text{A}
!RUN <RETURN>
```

(result)

```
0000 8000
0001 8001
...
...
00FF 80FF
```
HEX>

Description  
HEX> is a string function that assigns a string variable value in hexadecimal.

Type  
Statement

Syntax  
$A = \text{HEX}> \text{exp}$

where: exp is any valid expression.

Abbreviation  
None

See Also  
HEX$

Example 1  
To assign the ASCII characters "ABCDE" to string A, enter:

```
10 $A=\text{HEX}>4142434445
20 \text{PRINT} $A
!RUN <\text{RETURN}>
```

(result)  
ABCDE

Example 1  
The program below, in SIMP/L HDLC, sends a Restart packet from the Chameleon. The HEX> function stores the packet in a string.

```
10 $A = \text{HEX}>1000FB00
20 \text{BUILD} $A
30 \text{TRAN}
!RUN <\text{RETURN}>
```
### HLF

**Description**  
The HLF command causes text to be displayed in double intensity (highlight) on the Chameleon screen. Use the NRM command to cancel this and all other special display effects.

**Type**  
Statement or Direct

**Syntax**  
HLF

**Abbreviation**  
None

**See Also**  
NRM, HLFUND

**Example 1**  
This example causes a message to be displayed in double intensity on the screen.

```
10 HLF
20 PRINT "THIS IS BRIGHT"
30 NRM
!RUN <RETURN>
```

(result)  
THIS IS BRIGHT (appears in double intensity)
HLFUND

Description
The HLFUND command causes text to be displayed in double intensity (highlight) and underlined on the Chameleon screen. Use the NRM command to cancel this and all other special display effects.

Type
Statement or Direct

Syntax
HLFUND

Abbreviation
None

See Also
NRM, HLF

Example 1
This example displays a message in double intensity, underlined text.

10 HLFUND
20 PRINT "I'M BRIGHT"
30 NRM
!RUN <RETURN>

(result) I'M BRIGHT (appears in double intensity and underlined)
IF

Description
The IF commands allows program flow to be changed based on a decision. The IF command contains an expression and a command. The expression is evaluated as a condition. If the expression is true, the command is executed. If the expression is false, the command is ignored and the next line is executed.

Note: The word THEN is not used. String functions cannot be used directly inside an IF statement.

Type
Statement

Syntax
IF x op y command
where: x and y are numeric variables
op is a logical or arithmetic operator (=, # >, <, > =, < = AND OR)
command is the command to execute if the statement is true

Abbreviation
None

See Also
Arithmetic operators, logical operators, read-only variables

Example 1
This example tests a string value and prints a message if the test is true.

10 $A = "ABCD"
20 IF $A = "ABCD" PRINT "TRUE"
!/RUN <RETURN>

(result) TRUE
Example 2

This example determines if strings are equal and prints a message with the result.

10 $A = "ABCD"
20 $B = "ABC"
30 $C = "EFGH"
40 IF $A = $B GOTO 70
50 PRINT "$A IS NOT EQUAL TO $B"
60 GOTO 80
70 PRINT "$A IS EQUAL TO $B"
80 IF $A = $C GOTO 110
90 PRINT "$A IS NOT EQUAL TO $C"
100 GOTO 120
110 PRINT "$A IS EQUAL TO $C"
120 STOP
!RUN <RETURN>

(result) $A IS NOT EQUAL TO $B
        $A IS NOT EQUAL TO $C

Example 3

This example evaluates a variable using two IF statements.

10 A = 3
20 IF A # 3 GOTO 50
30 PRINT "A = 3"
50 IF A < 4 PRINT "A IS LESS THAN FOUR"
100 STOP
!RUN <RETURN>

(result) A = 3
        A IS LESS THAN FOUR
INKEY$$

Description
INKEY$$ is a string function that assigns the next character typed on the keyboard to a string variable. The INKEY$$ function assigns a value of zero if no key has been typed.

This function differs from the READ and $INPUT commands because it does not stop program execution until a key is pressed.

Note: To halt program execution while the program is waiting for input, press CTRL C.

Type
Statement

Syntax
$A = INKEY$$

Abbreviation
None

See Also
$INPUT

Example
This example prints the message TYPE A LETTER repeatedly until a character is typed. When a character is typed, it is displayed and then the TYPE A LETTER message is displayed repeatedly.

10 PRINT "TYPE A LETTER" Displays a message.
20 $A = INKEY$$ Check for keyboard input.
30 IF LEN($A) = 0 GOTO 10 If no key pressed, displays the message again.
40 PRINT $A Displays the character that is input.
50 GOTO 10 Displays the message again.
!RUN <RETURN>
INPUT

Description
The INPUT command stores keyboard input in a variable. It allows you to create screen prompts and stores the resulting keyboard input.

Note: To halt program execution while the program is waiting for input, press CTRL C.

Type
Statement

Syntax
INPUT "prompt"x

INPUT "prompt",x

where: prompt is the text that you want displayed on the screen (optional) and x is the variable that stores the keyboard input resulting from the prompt. In the second syntax form, the comma between the prompt and the variable displays the variable name following the prompt. A colon is automatically displayed at the end of the prompt.

Abbreviation
IN.

See Also
INKEY$

Example 1
10 INPUT "Enter your age "A
!RUN <RETURN>

(result) Enter your age:

Example 2
10 INPUT "Enter your age ",A
!RUN <RETURN>

(result) Enter your age A:

Example 3
10 INPUT "Enter any 3 numbers ",A,B,C
!RUN <RETURN>

(result) Enter any 3 numbers A:

The operator enters a number and presses return. He is then prompted for a second number. After entering a second number, he is prompted for a third number. Note that in this case, the variables A, B, and C are displayed in the prompt because they are separated by commas in the INPUT command.
You can also display multiple prompts using a single input command. For example, to prompt for the month, day and year of the user's birthday, you could enter:

```
10 INPUT "Enter month of birth"M,"Day of birth"D,"Year of birth"Y

!RUN <RETURN>
```

(result) Enter month of birth:

The user enters the number of the month and presses return. He is then prompted for the day and then the year. The values are stored in the variables M, D and Y.

If the response to the prompt is invalid, a Syntax error message is displayed.
$INPUT

<table>
<thead>
<tr>
<th>Description</th>
<th>$INPUT is a string function that enables you to assign a string variable from the keyboard.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement</td>
</tr>
<tr>
<td>Syntax</td>
<td>$INPUT $A</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>None</td>
</tr>
<tr>
<td>See Also</td>
<td>INKEY$, INPUT</td>
</tr>
</tbody>
</table>

**Example**

This program halts for input. When the user types a response and presses RETURN, the response is displayed on the screen

```
10 PRINT "ENTER YOUR NAME BELOW"
20 $INPUT $A
30 PRINT $A
!RUN  <RETURN>

(result)        ENTER YOUR NAME BELOW
                :ernie   <RETURN>
                ernie
```
### INS

**Description**  The INS command inserts a blank line on the Chameleon screen.

**Type**  Statement

**Syntax**  INS

**Abbreviation**  None

**See Also**  DEL, ERAEOL, ERAEOS, XYPLOT

**Example**  This example displays text on line 0 (top) of the Chameleon screen and then inserts a line 22 times. This causes the messages to move toward the bottom of the screen.

```
10 CLS  Clear the screen.
20 XYPLOT(0,0)  Positions cursor on top line of screen.
30 PRINT "HELLO"  Prints message on line 0. Note that there is a blank space between the opening quotation mark and the II in HELLO.
40 XYPLOT(0,0)  Positions cursor on line 0 of screen.
50 FOR A = 1 TO 22  Sets up loop to repeat 22 times.
60 INS  Inserts a line.
70 NEXT A  Increments loop counter.

!RUN <RETURN>
```
INSTR

Description  INSTR is a string function that returns the offset (position) of a substring within the main string.

Type  Statement

Syntax  \[ x = \text{INSTR}(\text{str1}, \text{str2}) \]

where: \( \text{str1} \) is the main string and \( \text{str2} \) is the substring. If the substring does not exist within the main string, INSTR is zero.

Abbreviation  None

See Also  MID$

Example  This example tests for the offset of DEF in the string ABCDEFGHIJK and prints the result.

10 \$A = "ABCDEFGHIJK"
20 \$B = "DEF"
30 A=INSTR($A,$B)
40 PRINT A
!RUN  <RETURN>

(result)  4

The string DEF starts with the fourth character $A, assigning the value of 4 to A.
## KILL

**Description**  
The KILL command deletes a specified file from either the hard or floppy disk drive.

**Type**  
Direct

**Syntax**  
- `KILL "name", x`
- `KILL "A: name", x`
- `KILL "B: name", x`

$A = "name"

KILL $A, x

*where: name is the filename and x is a letter specifying one of the following file types:*

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Program</td>
</tr>
<tr>
<td>T</td>
<td>Trace</td>
</tr>
<tr>
<td>M</td>
<td>Mnemonic table</td>
</tr>
<tr>
<td>D</td>
<td>Data</td>
</tr>
<tr>
<td>S</td>
<td>Setup (parameter)</td>
</tr>
<tr>
<td>F</td>
<td>Function key definition</td>
</tr>
<tr>
<td>A</td>
<td>All types</td>
</tr>
</tbody>
</table>

When you use the A option to delete all file types, message confirm either that a certain file type was not found, or that it was deleted.

**Abbreviation**  
K.

**See Also**  
FILES

**Example 1**  
This example deletes the program file named TEST23 from the hard disk drive.

```basic
KILL "TEST23",P
```

**Example 2**  
This example deletes the function key file names FKEYS1 from the floppy disk drive.

```basic
KILL "B:FKEYS1",F
```
LEFT$ is a string function that assigns a specified number of characters from the left end of one string to another string.

You cannot use the syntax PRINT LEFT$ to print a result. You must assign values to the string, as shown in Example 2 below.

Statement $A = \text{LEFT$}($x,exp)$

where: $x$ is the string that contains the characters and exp defines the number of characters from the left of $x$ to be assigned to $A$.

Example 1
This example prints the four leftmost characters of a string.

```
10 $A = "ABCDEFHJI"
20 $B = \text{LEFT$}($A,4)$
30 \text{PRINT }$B$
40 !\text{RUN }<\text{RETURN}>
```

(result) ABCD

Example 2
This program uses a FOR loop to assign and print string A using the LEFT$ function.

```
10 $A = "HELLO"
20 \text{FOR } A = 1 \text{ TO } 5
30 $B = \text{LEFT$}($A,A)$
40 \text{PRINT }$B$
50 !\text{NEXT } A
60 !\text{RUN }<\text{RETURN}>
```

(result) H
    HE
    HEL
    HELL
    HELLO
# LEN

**Description**
LEN is a string function that returns the length of a string variable.

**Type**
Statement or Direct

**Syntax**

```
A = LEN($x)
```

where: $x$ is a string variable and $A$ is the numeric variable assigned the length of $x$.

**Abbreviation**
None

**See Also**
MID$, RIGHT$, LEFT$

**Example**
This example assigns a string to a variable and then displays the length of the string.

```
10 $A = "ABCD"
20 A = LEN($A)
30 PRINT A
!RUN <RETURN>
```

(result) 4
## LET

### Description
The LET command assigns values to numeric or string variables. LET is optional and may be omitted.

### Type
Statement or Direct

### Syntax

```plaintext
LET x = exp

LET $A = "xxx"
```

*where*: x is numeric variable and exp is a valid expression. An expression can consist of arithmetic or logical expressions, functions, or mnemonics. $A is a string variable.

An ampersand symbol (&) preceding a number assigns a hexadecimal value.

### Abbreviation
None

### See Also
Arithmetic operators

### Example
This example assigns values of numeric variables, adds the variables, and displays the result.

```plaintext
10 LET X=10
20 LET Y=15
30 LET Z = X+Y
40 LET $A = "THE ANSWER IS: "
50 PRINT $A, Z
!RUN < RETURN>
```

(result): THE ANSWER IS 25

Optionally, this program could omit the use of LET, as shown below, with the same result.

```plaintext
10 X=10
20 Y=15
30 Z = X+Y
40 $A = "THE ANSWER IS: "
50 PRINT $A, Z
!RUN < RETURN>
```
### LFILES

**Description**  
The LFILES command prints the names of the Chameleon BASIC files found on a specified disk to the printer or remote device. The LFILES command prints:

- Program Files
- Trace Files
- Mnemonic Table Files
- Data Files
- Setup Files
- Function Key Files

**Type**  
Direct

**Syntax**  

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFILES</td>
<td>Prints a list of files on drive A (hard disk).</td>
</tr>
<tr>
<td>LFILES A</td>
<td>Prints a list of files on drive A (hard disk).</td>
</tr>
<tr>
<td>LFILES B</td>
<td>Prints a list of files on drive B (floppy disk).</td>
</tr>
</tbody>
</table>

**Abbreviation**  
LF.

**See Also**  
FILES

**Example**  
This example lists the files on the floppy disk drive.

```
LFILES B
(result)  Program Files
        : PROGA PROGB PROGC PROGD PROGE
Trace Files
Mnemonic Table Files
Data Files
        : DATA1 DATA2
Setup Files
        : DEFAULT SETUP1 SETUP2 SETUP3
Function Key Files
```
LFLIST

Description  The LFLIST command sends the list of current function key assignments to a printer or remote device.

Type  Statement or Direct

Syntax  LFLIST

Abbreviation  LFL.

See Also  FDEFINE, FLIST, FSAVE

Example  !LFLIST

(result)  KEY 1 = MLIST'FILES'FLIST'  
KEY 2 = COUPLER=PARA1'AUTO'
KEY 3 = ?"GOOD MORNING"'
KEY 4 = A+B/2'
KEY 5 = LLIST'
KEY 6 = LOAD "B:TEST1''RUN''
KEY 7 = FLUSH'NEW'AUTO'
KEY 8 = $A="HELLO''
KEY 9 = TRAN'GOTO10'
KEY 10 = RESEQ'LIST'

Note that the tildes that are used to define the keys in the FDEFINE command are not displayed with FLIST.Example
LIST

Description
The LIST command displays all or specified lines of a Chameleon BASIC program. To stop the list, use CTRL S. To resume, press any key. Use the ESCape key to abort it entirely.

Type
Direct

Syntax
 LIST  Lists the entire program.
 LIST x  Lists program from line x to end.
 LIST x,y  Lists program from line x to line y.
 LIST ,y  Lists program from beginning to line y.

Abbreviation
L.

See Also
LLIST

Example 1
To list the entire program in memory, enter:
 LIST

Example 2
To list lines 40 - 60 of the program in memory, enter:
 LIST 40,60

Example 3
To list lines 1 - 60 of the program in memory, enter:
 LIST ,60
**LLIST**

**Description**  
The LLIST command outputs all or specified lines of the program in memory to a printer or remote device.

**Type**  
Direct

**Syntax**

- LLIST  
  Prints the entire program.
- LLIST x  
  Prints program from line x to end.
- LLIST x,y  
  Prints program from line x to line y.
- LLIST ,y  
  Prints program from beginning to line y.

**Abbreviation**  
 LL.

**See Also**  
 LIST

**Example 1**  
To output the entire program in memory to the printer, enter:  
LLIST

**Example 2**  
To output lines 40 - 60 of the program in memory to the printer, enter:  
LLIST 40,60

**Example 3**  
To output line 60 of the program in memory to the printer, enter:  
LLIST ,60
LMLIST

**Description**  The LMLIST command outputs the mnemonic table in memory to a printer or remote device.

**Type**  Direct

**Syntax**  LMLIST

**Abbreviation**  LML.

**See Also**  MLIST, MLOAD, MSAVE

**Example**  To output the mnemonic table in memory to the printer, enter:

LMLIST
# LOAD

**Description**
The LOAD command loads a BASIC program into memory.

**Type**
Direct

**Syntax**
- LOAD "name"
- LOAD "A:name"
- LOAD "B:name"
- $A = "name"
- LOAD $A

where: name is the program filename.

**Abbreviation**
LO.

**See Also**
CHAIN, RUN, NEW

**Example 1**
This example loads the BASIC program named TEST1 into memory from the hard disk.

```
LOAD"TEST1"
```

**Example 2**
This example loads the BASIC program named ANALYZE into memory from the floppy disk.

```
LOAD"B:ANALYZE"
```
**LPRINT**

<table>
<thead>
<tr>
<th>Description</th>
<th>The LPRINT command sends the contents of the trace buffer to a printer or remote device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>LPRINT</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>LTP.</td>
</tr>
<tr>
<td>See Also</td>
<td>TPRINT, TSAVE, TLOAD</td>
</tr>
<tr>
<td>Example</td>
<td>LPRINT</td>
</tr>
</tbody>
</table>
### MENU

<table>
<thead>
<tr>
<th>Description</th>
<th>The MENU command exits the simulator and returns to the Chameleon main menu so that you can change simulation languages or switch to the Monitor mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>MENU</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>MEN.</td>
</tr>
<tr>
<td>See Also</td>
<td>SETUP</td>
</tr>
<tr>
<td>Example</td>
<td>MENU</td>
</tr>
</tbody>
</table>
**MERGE**

**Description**
The MERGE command combines a specified program source file with the program already in memory. The two programs are merged as if the merged one were entered from the keyboard.

This command is especially useful for altering only the main code or just the subroutines of two similar programs.

If the program you are merging has line numbers that do not exist in the program in memory, these lines will be added. Lines with identical line numbers will overwrite the original program. For this reason, use caution when merging programs with GOTO and GOSUB statements.

**Type**
Direct

**Syntax**
MERGE "name"
MERGE "A:name"
MERGE "B:name"

$A = "name"
MERGE $A

**Abbreviation**
M.

**Example**
Program 1 is in a file called PROG1 and is already in memory as shown below:

10 REM PROG 1 LINE 10
15 REM PROG 1 LINE 15
30 REM PROG 1 LINE 30

Program 2 is in a file called PROG2 on the B drive. It contains three lines of code as shown below. Note that both PROG1 and PROG2 contain lines numbered 30.

8 REM PROG 2 LINE 8
16 REM PROG 2 LINE 16
30 REM PROG 2 LINE 30

To merge PROG1 and PROG2, enter:

MERGE "B:PROG2"

(result) 8 REM PROG 2 LINE 8
10 REM PROG 1 LINE 10
15 REM PROG 1 LINE 15
16 REM PROG 2 LINE 16
30 REM PROG 2 LINE 30
### MIDS$

**Description**

MIDS is a string function that assigns one or more characters from the middle of a string to a string variable.

Note: You cannot use the syntax `PRINT MIDS` to print a result. You must assign the string to a variable, as shown in the example below.

**Type**

Statement

**Syntax**

\[ \$A = \text{MIDS}(\$x, \text{exp1}, \text{exp2}) \]

where: \text{exp1} defines the position of the first character and \text{exp2} defines the number of characters in \$x to assign to \$A.

**Abbreviation**

None

**See Also**

LEFT$, RIGHT$, INSTR

**Example**

10 \$A = "ABCD"
20 \$B = \text{MIDS}(\$A, 2, 1)
30 \text{PRINT} \$B

!RUN < RETURN>

(result) \$B

3.5-69 6/11/90
MLIST

Description
The MLIST command displays the mnemonic table in memory. Refer to Section 3.2 for a general description of the use of mnemonic tables. For information about a table for a specific language or protocol, refer to the appropriate chapter for the protocol you are using.

To stop the mnemonic table list, type CTRL S. To resume the list, press any key. Press ESC to abort the listing.

Type
Statement or Direct

Syntax
MLIST

Abbreviation
ML.

See Also
LMLIST, MLOAD, MSAVE, DEFINE

Example
MLIST
MLOAD

**Description**
The MLOAD command loads a mnemonic table from disk to memory.

**Type**
Statement or Direct

**Syntax**
- MLOAD"name"
- MLOAD"A:name"
- MLOAD"B:name"
- $A = "name"
- MLOAD $A

*where: name is the mnemonic table filename.*

**Abbreviation**
MLO.

**See Also**
MSAVE, MLIST, DEFINE

**Example 1**
To load the mnemonic table file named TABLE1 into memory from the hard disk drive, enter:

```
MLOAD"TABLE1"
```

**Example 2**
To load the mnemonic table file named TABLE2 into memory from the floppy disk drive, enter:

```
MLOAD"B:TABLE2"
```
**MSAVE**

**Description**
The MSAVE command saves the mnemonic table in memory to disk.

**Type**
Direct

**Syntax**
- `MSAVE"name"`
- `MSAVE"A:name"`
- `MSAVE"B:name"`
- `$A = "name"
  MSAVE $A`

*where*: `name` is the mnemonic table filename.

**Abbreviation**
MSA.

**See Also**
MLOAD, MLIST, LMLIST, DEFINE

**Example 1**
To save the mnemonic table in memory to the hard disk drive with the filename TABLE1, enter:

`MSAVE"TABLE1"`

**Example 2**
To save the mnemonic table in memory to the floppy disk drive with the filename TABLE2, enter:

`MSAVE"B:TABLE2"`
NEW

Description

The NEW command deletes the program in memory so that you can enter a new program.

Caution! NEW loses the original program completely. Use the SAVE command to save the current program to disk, before you enter the NEW command.

Type

Direct

Syntax

NEW

Abbreviation

N.

See Also

SAVE

Example

NEW
# NEXT

<table>
<thead>
<tr>
<th>Description</th>
<th>The NEXT command increments the counter in a FOR loop. The amount of the increment is determined by the STEP in a FOR statement. If a STEP is not specified, the default of one is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement</td>
</tr>
<tr>
<td>Syntax</td>
<td>NEXT x</td>
</tr>
<tr>
<td></td>
<td><em>where:</em> x is the variable that represents the loop counter in the FOR statement.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>NEX.</td>
</tr>
<tr>
<td>See Also</td>
<td>FOR</td>
</tr>
<tr>
<td>Example</td>
<td>This example prints odd numbers between 1 and 10.</td>
</tr>
<tr>
<td></td>
<td>10 FOR X = 1 TO 10 STEP 2</td>
</tr>
<tr>
<td></td>
<td>20 PRINT X</td>
</tr>
<tr>
<td></td>
<td>30 NEXT X</td>
</tr>
<tr>
<td></td>
<td>!RUN &lt;RETURN&gt;</td>
</tr>
<tr>
<td></td>
<td>(result) 1</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>
NRM

Description  The NRM command cancels special display effects (blinking, underline, double intensity) and returns the Chameleon to normal display.

Type  Statement or Direct

Syntax  NRM

Abbreviation  None

See Also  BLK, BLKREV, BLKUND, HLF, HLFUND, CLS

Example  This example displays the message I'M BLINKING in blinking text and then the message NOW I'M NOT BLINKING in normal display.

```
10 BLK
20 PRINT "I'M BLINKING"
30 NRM
40 PRINT "NOW I'M NOT BLINKING"
!RUN <RETURN>
```
OPEN

Description
The OPEN command opens a disk data file so that data can be read, written, or appended to the file. You can have one file for input and one file for output opened at the same time.

Caution
If you open a new file in output mode when there is an existing file of the same name on the disk, the original file is deleted from the disk and is replaced with the new data.

Type
Statement or Direct

Syntax
OPEN "I","name" Opens a file for input so that data can be read from the file.
OPEN "O","name" Opens a new file for output so that data can be written to the file.
OPEN "A","name" Opens an existing file for output so that data can be added to it.

where: name is the data file name.

Note: When calling a sub-program, the system automatically closes all files. This occurs only when calling a sub-program, and not when exiting a program.

Abbreviation
O.

See Also
CLOSE, READ, WRITE
Example

The example below opens a new data file, adds a record, then closes the file.

10 A = 0
20 $A = "THIS IS RECORD #"
30 OPEN "O", "B:TEST"

40 $B = $A+HEX$(A)
50 WRITE $B

60 CLOSE
70 FOR A = 1 TO 4000
80 GOSUB 160
90 PRINT %A
100 $B = $A+HEX$(A)
110 OPEN "A", "B:TEST"
120 WRITE $B
130 CLOSE
140 NEXT
150 STOP;
160 OPEN "I", "B:TEST"
170 READ $Z
180 IF EOF GOTO 210
190 PRINT $Z
200 GOTO 170
210 CLOSE
220 RETURN

Initializes record counter.
Creates file named TEST on drive B.
Writes first record which will be 0000.
Closes file.
Executes following lines 4000 times.
Subroutine to read from file.
Prints "FOR" variable in hex.
Assigns incremented number to the string.
Opens existing file for append.
Writers a record.
Closes file.
Continues loop.
Opens file for input.
Reads from the file.
If end of line, closes file.
Prints $Z.
Reads $Z.
Closes file.
PRINT

Description The PRINT command prints a string, expression, or value of a variable on the screen.

Type Statement or Direct

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT &quot;string&quot;</td>
<td>Prints the string.</td>
</tr>
<tr>
<td>PRINT $A</td>
<td>Prints the string variable.</td>
</tr>
<tr>
<td>PRINT x</td>
<td>Prints the numeric variable.</td>
</tr>
<tr>
<td>PRINT %x</td>
<td>Prints the expression x in hex.</td>
</tr>
</tbody>
</table>

Options You can use the following print options.

- A comma (,) acts as a field separator
- A backslash (\) suppresses a line feed
- A semicolon (;) suppresses the carriage return

Numeric Format You can assign the number of spaces in which to print a numeric variable using the format symbol (#) followed by a valid expression. The valid expression can be a number, numeric variable, or other numeric expression that defines the number of spaces to assign.

The default format specifies six spaces for each numeric variable. If the number of spaces is not adequate for the value, more spaces are added. In other words, short format specifications do not result in truncated values. The format specification is ignored for strings.

Abbreviations P. or ?

Example 1 This example prints a message.

```
PRINT "HELLO"
```

(result) HELLO

Example 2 This example prints a message in hex.

```
10 $A = "HELLO"
20 PRINT %$A
!RUN <RETURN>
```

(result) 68 65 6C 6C 6F
Example 3

This example prints a number several times.

```
10 A = 1234
20 FOR B = 3 TO 7
30 PRINT #B, A
40 NEXT B
!RUN <RETURN>
```

(result)  
1234  
1234  
1234  
1234  
1234
**READ**

**Description**
The READ command reads the next record from an input file into a string variable. The file must already be open for input.

**Type**
Statement

**Syntax**
READ $A

**Abbreviation**
REA.

**See Also**
OPEN, WRITE

**Example**

```
10 OPEN "I","DATA1"  \* Opens a file named Data1 for input.
20 READ $A  \* Reads a record from Data1 file into $A.
30 IF EOF GOTO 50  \* If at end of file, goes to line 50.
40 PRINT $A  \* If not at end of file, prints $A.
50 CLOSE  \* Closes the data file.
```

!RUN  <RETURN>
Description

The REC command transfers data from the acquisition buffer to the trace buffer. The frame length, status (CRC) and a timestamp are added to the beginning of each frame. Refer to Section 3.2 for a description of the buffers used in simulation.

The use of the REC command is protocol-specific. Refer to the appropriate chapter for a description of the syntax for the simulator you are using.
### REM

<table>
<thead>
<tr>
<th>Description</th>
<th>The REM commands enables you to place internal comments in your program file. The comments provide program documentation and are ignored by the Chameleon. They are not displayed on the screen when the program is executed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>REM comment</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>None</td>
</tr>
<tr>
<td>See Also</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>10 REM This routine defines mnemonics and 20 REM Receives a frame</td>
</tr>
</tbody>
</table>
## RESEQ

### Description

The RESEQ command re-numbers (re-sequences) the lines of the program in memory. If the optional expressions \{EXPR1\} and \{EXPR2\} are not used to set the re-numbering starting point and increment, re-numbering starts at line 10 and increases in increments of 10.

### NOTE:

RESEQ destroys the contents of any strings, arrays, and buffers in use. "Beginning Line Number" refers to the number assigned to the first program line, NOT to the original line number at which re-numbering begins.

Line numbers within GOTO and GOSUB statements will be changed only when numeric line numbers are used. For numeric expressions, the first number in the expression is used as the instruction and the remainder of the expression is ignored. For example, GOTO 2+10 is treated the same as GOTO 2, and not as GOTO 12.

Three possible messages indicating areas that may require manual numbering changes may appear while the system is re-numbering your program:

- **Number of lines in program** gives you the total number of lines that are in the program.
- **GOTO or GOSUB not resequenced in line #** indicates that a line number within a GOTO or GOSUB command could not be correctly re-sequenced. The message references the new line number of the GOTO or GOSUB command.
- **Non-existent line # found in line #** tells you that your program contained a command that referenced a line number that did not exist in the original numbering. If a non-existent line number is listed, it may exist in the program after it is re-sequenced as in the example below.

### Type

Direct

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESEQ</td>
<td>Re-sequences the program to begin with line number 10, in line number increments of 10.</td>
</tr>
<tr>
<td>RESEQ {EXPR1}</td>
<td>An optional expression for setting the number assigned to the first program line.</td>
</tr>
<tr>
<td>RESEQ {EXPR1} {EXPR2}</td>
<td>An optional expression for setting the value by which line numbers are incremented.</td>
</tr>
</tbody>
</table>
NOTE: If EXPR1 and EXPR2 are omitted, the default value for each RESEQ command is 10. Also, EXPR1 must be used if EXPR2 is used.

Abbreviation
RES.

See Also
AUTO, ERASE

Example
The original program is shown below:

```
5  GOTO 70
10  REM  ORIGINAL PROGRAM
11  REM  WITH ADDITIONAL LINES
20  FOR  A  =  1  TO  10
25  PRINT  A
40  NEXT  A
45  IF  A  =  10  GOTO  50
46  IF  A  =  1  GOTO  A + 4
47  PRINT  "ERROR IN LOOP"
50  STOP
```

To renumber the program to start with line 1000 in increments of 10, enter:

```
!RESEQ {1000,10}
```

The following messages are displayed:

```
Number of lines in program 10
Non-existent line 70 found in line 1000
GOTO or GOSUB not resequenced in line 1070
```

The program will be re-numbered as follows:

```
1000  GOTO 70
1010  REM  ORIGINAL PROGRAM
1020  REM  WITH ADDITIONAL LINES
1030  FOR  A  =  1  TO  10
1040  PRINT  A
1050  NEXT  A
1060  IF  A  =  10  GOTO  100
1070  IF  A  =  1  GOTO  A + 4
1080  PRINT  "ERROR IN LOOP"
1090  STOP
```
# RETURN

**Description**  
The RETURN command returns program control from a subroutine called by a GOSUB command to the statement following the GOSUB.

**Type**  
Statement

**Syntax**  
RETURN

**Abbreviation**  
RET.

**See Also**  
GOSUB, GOTO

**Example**  
```
10 PRINT "HELLO"
20 GOSUB 50
30 PRINT "HELLO AGAIN"
40 STOP
50 PRINT "GOODBYE"
60 RETURN
!RUN <RETURN>
```

(result)  
HELLO
GOODBYE
HELLO AGAIN
REV

Description
The REV command displays text in reverse video on the
Chameleon screen. The NRM command cancels this and all
other display effects and returns the screen to normal mode.

Type
Statement or Direct

Syntax
REV

Abbreviation
None

See Also
NRM, BLKREV

Example 1
10 REV
20 PRINT "I'M IN REVERSE VIDEO"
30 NRM
40 PRINT "I'M BACK TO NORMAL"
!RUN <RETURN>

(result) I'M IN REVERSE VIDEO (reverse video)
I'M BACK TO NORMAL (normal text display)
## REVHLF

### Description

The REVHLF command displays text in reverse video in double intensity on the Chameleon screen. The NRM command cancels this and all other display effects and returns the screen to normal mode.

### Type

Statement or Direct

### Syntax

REVHLF

### Abbreviation

None

### See Also

NRM, BLKREV, REV

### Example 1

```
10 REVHLF
20 PRINT "I'M IN HIGHLIGHTED REVERSE VIDEO"
30 NRM
40 PRINT "I'M BACK TO NORMAL"
!RUN  <RETURN>
```

(result)  

```
I'M IN HIGHLIGHTED REVERSE VIDEO
I'M BACK TO NORMAL
```
## REVUND

### Description
The REVUND command displays text in reverse video and underlined on the Chameleon screen. The NRM command cancels this and all other display effects and returns the screen to normal mode.

### Type
Statement or Direct

### Syntax
REVUND

### Abbreviation
None

### See Also
NRM, BLKREV, REV, REVHIF

### Example 1
```
10 REVUND
20 PRINT "I'M IN UNDERLINED REVERSE VIDEO"
30 NRM
40 PRINT "I'M BACK TO NORMAL"
!RUN    <RETURN>

[result] I'M IN UNDERLINED REVERSE VIDEO
         I'M BACK TO NORMAL
```
# RIGHT$  

**Description**
RIGHT$ is a string function that assigns a specified number of characters from the right end of one string to another string.

*Note:* You cannot use the syntax `PRINT RIGHT$` to print a result. You must assign the string to a variable, as shown in Example 1 below.

**Type**
Statement or Direct

**Syntax**

```plaintext
$A = RIGHT$(x,exp)
```

*where:* $x$ is the string that contains the characters, `exp` defines the number of characters from the right to be assigned to `$A`.

**Abbreviation**
None

**See Also**
LEFT$, MID$

## Example 1
This example assigns a string to a variable and then prints the rightmost four characters.

```plaintext
10 $A = "ABCDEFGH"
20 $B = RIGHT$(A,4)
30 PRINT $B
!RUN <RETURN>
```

(result) EFGH

## Example 2
This example assigns a string to a variable and then prints parts of it several times.

```plaintext
10 $A = "HELLO"
20 FOR A = 1 TO 5
30 $B = RIGHT$(A,A)
40 PRINT $B
50 NEXT A
!RUN <RETURN>
```

(result) 0
LO
LLO
ELLO
HELLO
**RND**

**Description**
The RND command returns a random number between zero and a specified number.

**Type**
Statement or Direct

**Syntax**
RND(x)

where: a random number between zero and x is returned. The parentheses are optional.

**Abbreviation**
R.(x)

**See Also**
Arithmetic operators.

**Example**

10 A = RND(100)  
20 PRINT A  
1 RUN <RESULT> 

(result) 55 (or any random number between 0 and 100)
RUN

Description
The RUN command executes a program in memory, starting at its first line.

Type
Direct

Syntax
RUN

Abbreviation
RU.

See Also
CHAIN, CALL

Example
This example loads a program called PROG1 from the hard disk and then executes it.

!LOAD "PROG1"
!RUN

Note that you could also use the CHAIN command to load and run a program with a single command. For example, you could accomplish the same as the above example, by entering:

!CHAIN "PROG1"
SAVE

Description The SAVE command saves the program in memory to disk.

Type Direct

Syntax
SAVE "name"
SAVE "A:name"
SAVE "B:name"
$A = "name"
SAVE $A

where: name is the program filename.

Abbreviation SA.

See Also LOAD, KILL, FILES

Example 1 To save the program in memory to a file named PROG1 on the hard disk drive, enter:
SAVE"A:PROG1"

Example 2 To save the program in memory to a file named TESTPROG on the floppy disk drive, enter:
SAVE"B:TESTPROG"

Example 3 To save the program in memory to a file named TESTPROG on the floppy disk drive, enter:
$A = "B:TESTPROG"
SAVE $A
SET

Description

SET sets a specified physical interface signal to either logical 1 or 0. The SET command is not available in SIMPL or FRAMEM DMI.

Type

Statement or Direct

Syntax

SET xxx = y

where: y is a 1 or 0 and xxx is the physical interface NAME (see table below). Note that the physical interface signals that you can set depends on whether the Chameleon is configured as a DCE or DTE.

<table>
<thead>
<tr>
<th>NAME</th>
<th>FUNCTION</th>
<th>CCITT</th>
<th>RS232 Pin</th>
<th>SET BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTS</td>
<td>Clear To send</td>
<td>106</td>
<td>5</td>
<td>DCE</td>
</tr>
<tr>
<td>DSR</td>
<td>Data Set Ready</td>
<td>107</td>
<td>6</td>
<td>DCE</td>
</tr>
<tr>
<td>DCD</td>
<td>Data Carrier Detect</td>
<td>109</td>
<td>8</td>
<td>DCE</td>
</tr>
<tr>
<td>RI</td>
<td>Ring Indicator</td>
<td>125</td>
<td>22</td>
<td>DCE</td>
</tr>
<tr>
<td>SDCD</td>
<td>Secondary Data Carrier Detect</td>
<td>122</td>
<td>12</td>
<td>DCE</td>
</tr>
<tr>
<td>DTR</td>
<td>Data Terminal Ready</td>
<td>108</td>
<td>20</td>
<td>DTE</td>
</tr>
<tr>
<td>RTS</td>
<td>Request To send</td>
<td>105</td>
<td>4</td>
<td>DTE</td>
</tr>
</tbody>
</table>

Table 3.5-1: SET Physical Interface Options

Abbreviation

None

See Also

TEST

Example

10 SET RTS=1
20 TEST CTS=0 GOTO 20
30 TRAN
40 SET RTS=0
! RUN <RETURN>

Sets the Request To Send to 1.
Waits until DCE sets CTS = 1.
Transmits the data.
Disconnects the link at the physical level by returning RTS to 0.
SETUP

Description: The SETUP command exits simulation programming mode (!) and returns to the parameter set-up menu for a particular simulation language.

Refer to Section 2.2 for a general description of parameter set-up menus. For a description of a specific menu, refer to the simulation language chapter you are interested in.

*The SETUP command is not available in FRAMEM DMI.*

Type: Direct

Syntax: SETUP

Abbreviation: SE.

See Also: MENU

Example: SETUP
SIZE

Description  SIZE is a read-only variable that returns the size of free program area in bytes. There is a maximum of 8K bytes of programming memory for you to use for your BASIC programs. The SIZE variable enables you to determine how much memory is available.

Type  Variable

Syntax  (See examples below.)

Abbreviation  SI.

See Also  NEW

Example 1  To display the number of bytes of programming memory available, enter:

PRINT SIZE

Example 2  To test available memory and display a message when it is below 1000 bytes, enter:

IF SIZE < 1000 PRINT "GETTING LOW ON MEMORY"
STOP

<table>
<thead>
<tr>
<th>Description</th>
<th>The STOP command terminates program execution whenever encountered.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement</td>
</tr>
<tr>
<td>Syntax</td>
<td>STOP</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>ST.</td>
</tr>
<tr>
<td>See Also</td>
<td>EXIT</td>
</tr>
<tr>
<td>Example</td>
<td>This example prints a list of numbers, and then executes a stop command to end the program.</td>
</tr>
</tbody>
</table>

```
10 FOR A = 1 TO 10
20 PRINT A
30 IF A > 5 GOTO 50
40 NEXT A
50 STOP
!RUN  <RETURN>
```

<table>
<thead>
<tr>
<th>(result)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>


TEST

Description
The TEST command tests the level of an interface signal for logical 1 or 0. If true, the statement following TEST is executed. *The TEST command is not available in SIMP/L or FRAMEM DMI.*

Type
Statement or Direct

Syntax
**TEST** xxx = y statement

where: y is a logical one or zero, xxx is one of the physical interface NAMES in the table below, and statement is any valid program statement that you want to execute when the TEST condition is true.

You cannot use the not equal (#) in a TEST command.

Note that the physical interface signals you can test depend on whether the Chameleon is configured as a DCE or DTE.

<table>
<thead>
<tr>
<th>NAME</th>
<th>FUNCTION</th>
<th>CCITT</th>
<th>RS232 Pin</th>
<th>TESTED BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTS</td>
<td>Clear To send</td>
<td>106</td>
<td>5</td>
<td>DTE</td>
</tr>
<tr>
<td>DSR</td>
<td>Data Set Ready</td>
<td>107</td>
<td>6</td>
<td>DTE</td>
</tr>
<tr>
<td>DCD</td>
<td>Data Carrier Detect</td>
<td>109</td>
<td>8</td>
<td>DTE</td>
</tr>
<tr>
<td>RI</td>
<td>Ring Indicator</td>
<td>125</td>
<td>22</td>
<td>DTE</td>
</tr>
<tr>
<td>SDCD</td>
<td>Secondary Data Carrier Detect</td>
<td>122</td>
<td>12</td>
<td>DTE</td>
</tr>
<tr>
<td>DTR</td>
<td>Data Terminal Ready</td>
<td>108</td>
<td>20</td>
<td>DCE</td>
</tr>
<tr>
<td>RTS</td>
<td>Request To send</td>
<td>105</td>
<td>4</td>
<td>DCE</td>
</tr>
</tbody>
</table>

Table 3.5-2: TEST Physical Interface Options

Abbreviation
TE.

See Also
SET

Example
10 TEST RTS=0 GOTO 40
20 PRINT 'RTS = 1'
30 GOTO 10
40 PRINT 'RTS = 0'
50 GOTO 10
! RUN <RETURN>

If RTS is zero, go to line 40.
Otherwise print "RTS = 1."
Test again.
Print "RTS = 0."
Test again.

6/11/90
**TFREE**

**Description**
TFREE is a read-only variable that returns the length of the unused trace buffer in bytes. There is a maximum of 4K bytes of trace memory available for storing transmitted and received frames.

**Type**
Statement or Direct

**Syntax**
(See examples below.)

**Abbreviation**
TF.

**See Also**
FREE, SIZE

**Example 1**
To display the amount of trace buffer available, enter:

```
PRINT TFREE
```

**Example 2**
To test the available trace memory and print a message when there is less than 100 bytes available.

```
IF TFREE < 100 PRINT "BUFFER IS GETTING FULL"
```

**Example 3**
This example tests for sufficient space in the trace buffer for a received frame.

```
10 REC Receives a frame.
20 IF RXFRLEN=0 GOTO 10 If no frame received, try to receive again.
30 IF TFREE < RXFRLEN + 8 GOTO 100 If the trace buffer space available is less than the received frame length (RXFRLEN) and added information, goto line 100. The 8 bytes of added information are 5 bytes for time stamp, 1 byte for status, and 2 bytes for length.

100 ...
```
TIME

Description
TIME is a read-only variable that returns a specified byte of the system time in BCD digits.

Type
Statement or Direct

Syntax
TIME(x)

where: x is in the range 0 to 4 and specifies the unit of time, as follows:

0 - hours
1 - minutes
2 - seconds
3 - 1/100s seconds (.01)
4 - 1/10s of milliseconds (.0001)

Abbreviation
None

See Also
ATIMES$, TIME$

Example 1
To display the current hour according to the system clock, enter:

PRINT TIME(0)

Example 2
To display the current seconds according to the system clock, enter:

PRINT TIME(2)

Example 3
To execute a program at midnight (for example, to save traffic for an hour), enter:

10 IF TIME(0) = 0 CALL "SAVETRAF"
20 GOTO 10
TIMES$  

Description  TIMES$ is a string function that assigns the current time according to the internal realtime clock to a string variable. The display is in BCD in units of hours, minutes, seconds, hundredths of seconds, and tenths of milliseconds.

Type  Statement or Direct

Syntax  TIMES$

Abbreviation  None

See Also  ATIME$, TIME

Example 1  This example displays the time.

10 $A = TIMES$
20 PRINT %$A

(result)  10 53 23 23 10  (or current system time)

Example 2  The example below displays the time in hex.

5  $A = TIMES$  Stores time in $A.$
10 FOR X = 1 TO 5  Loops five times.
20 $B = MIDS($A,X,1)  Stores HH, MM, SS, HHS and TM in $B$ according to loop index.
30 PRINT %VAL($B)  Prints hexadecimal values.
40 NEXT X  Increments loop index X.
!RUN  <RETURN>

3.5-100  6/11/90
**TLOAD**

**Description**
The TLOAD command loads a trace file into memory. A trace stores the contents of the trace buffer. For general information about simulation buffers, refer to Section 3.2.

**Type**
Statement or Direct

**Syntax**
- TLOAD"name"
- TLOAD"A:name"
- TLOAD"B:name"

$$A = \text{"name"}$$
TLOAD $A$

*where: name is the trace filename.*

**Abbreviation**
TLO.

**See Also**
TSAVE, TPRINT, LPRINT

**Example 1**
This example loads the trace file named TRFILE1 from the hard disk drive into memory and then prints a copy of the trace file on the printer.

```
10 TLOAD"TRFILE1".
20 LPRINT
!RUN <RETURN>
```

**Example 2**
This example produces the same result as the first example, but specifies the trace file name using a string variable.

```
10 $A="TRFILE1"
20 TLOAD $A
30 LPRINT
!RUN <RETURN>
```
TPRINT

Description

The TPRINT command displays the contents of the trace buffer which stores the transmitted and received traffic (following a REC command). The format of the display is protocol-specific. Refer to the appropriate chapter for more information about the protocol you are using.

You must use a REC command in the program to store the received frames in the trace buffer. The trace does not necessarily display frames in the order that they are transmitted and received. It displays them according to TRAN and REC commands specified in your program.

You can stop the list using CTRL S. Resume the list with CTRL Q. Abort using the ESC key.

The trace buffer display is protocol-specific. The general format of the trace buffer display is as follows:

```
T = Transmitted frame
R = Received frame
\? = Bad CRC (frame overflow or abort)
Space = Good CRC

T R ? 00:00:00:00:00:00 xxxx ...

Time Stamp             Protocol-specific
Hours:Minutes:Seconds:Seconds: 1Seconds: 001 Seconds
- First byte of frame

1-4 digits indicating frame
length (right-justified)
```

Type

Statement or Direct

Syntax

TPRINT

Abbreviation

TP

See Also

TLOAD, TSAVE, LTPRINT

Example

(See protocol-specific chapters for examples.)
## TROFF

<table>
<thead>
<tr>
<th>Description</th>
<th>The TROFF command turns the program trace facility (debug mode) off.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>TROFF</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>TROF.</td>
</tr>
<tr>
<td>See Also</td>
<td>TRON</td>
</tr>
<tr>
<td>Example.</td>
<td>To set the debugging facility off and run the program in memory, enter:</td>
</tr>
<tr>
<td></td>
<td>!TROFF</td>
</tr>
</tbody>
</table>
TRON

Description  The TRON command turns the program trace facility (debug mode) on. In program debug mode, the program is traced by line number. Each time a new line is executed, the line number is displayed in brackets.

Type  Direct

Syntax  TRON

Abbreviation  T.

See Also  TROFF

Example  When the following program is run:

10 FOR A = 1 TO 5
20 PRINT A
30 NEXT A
!RUN  <RETURN>

(result)  Line#         Value of A
          [10][20]  1
          [30][20]  2
          [30][20]  3
          [30][20]  4
          [30][20]  5
TSAVE

Description
The TSAVE command saves the contents of the trace buffer to a trace file.

Type
Statement or Direct

Syntax
TSAVE "name"
TSAVE "A:name"
TSAVE "B:name"

$A = "name"
TSAVE $A

where: $A is the trace filename.

Abbreviation
TSA.

See Also
TLOAD, TPRINT, LTPRINT

Example
This example saves received traces with a length less than 1000 to a file named NEWTRACE on the hard disk drive.

10 $A = "NEWTRACE"
20 REC
30 IF TFREE < 1000 GOTO 50
40 GOTO 20
50 TSAVE $A
! RUN <RETURN>
UND

Description  The UND command displays text in underline on the Chameleon screen. Use the NRM command to cancel this and all other display effects and return to normal display.

Type        Statement or Direct

Syntax       UND

Abbreviation None

See Also    NRM, REVUND, BLKUND

Example 1  This example displays a message in underlined text and then displays a message in normal text.

10 UND
20 PRINT "HELP! I'M BEING UNDERLINED!"
30 NRM
40 PRINT "THANK YOU"
!RUN   <RETURN>

(result)   HELP! I'M BEING UNDERLINED!
           THANK YOU
VAL

Description
VAL is a string function that converts the first two characters of a string to their numeric form.

Type
Statement or Direct

Syntax
A = VAL($A)

where: $A is an ASCII string.

Abbreviation
None

See Also
CHR$

Example 1
This example converts the string value 12 to its integer equivalent, and then prints the hexadecimal equivalent of the integer variable A.

10 $A = "12"
20 A = VAL($A)
30 PRINT %A
!RUN <RETURN>

(result) 3132

Example 2
This example assigns the decimal equivalent of the rightmost three letters of $A to $B, and then prints the hex equivalent of $A and $B.

10 $A="HELLO"
20 $B=RIGHT$(A,3)
30 A=VAL($A)
40 B=VAL($B)
50 PRINT %A
60 PRINT %B
70 STOP
!RUN <RETURN>

(result) 4845
4C4C
WRITE

Description: The WRITE command writes a string variable to a data file opened for output.

Type: Statement

Syntax: WRITE $A

where: $A is the string variable written to the open data file.

Abbreviation: WR.

See Also: OPEN, CLOSE, EOF

Example: This example assigns a string to a string variable and then writes the string to a data file.

10 $A = "ABCDE"
20 OPEN"O",DATA1
30 WRITE $A
!RUN  <RETURN>
XYPLOT

Description
The XYPLOT command moves the cursor to a specified position on the Chameleon screen.

Type
Statement or Direct

Syntax
XYPLOT(y,x)

where: y is the y-coordinate (row) in the range 0 - 21 and x is the x-coordinate (column) in the range 0 - 79. Although the screen is 24 lines by 80 columns, you cannot use lines 22 and 23 because they are reserved for page banners and softkey strips. If you use 22 or 23 as the y coordinate, line 21 will be used.

The top left hand corner has the coordinates (0,0). The bottom right hand corner has the coordinates (21,79), as shown in the figure below.

```
     X = 0 to 79
  
     Y = 0 to 21

   0,0  0,79
  21,0  21,79
```

If you use 21 as the y coordinate, a line feed results, unless you suppress the line feed with a comma in the PRINT statement, as follows:

```
10 XYPLOT(21,0)
20 PRINT "HELLO",
```

Abbreviation
None

See Also
BLK, BLKREV, BLKUND, CLS, DEL, ERAEOL, ERAEOS, HLF, HLFUND, INS, NRM, REV, REVHLF, REVUND, UND
Example

This example prompts for user input and then displays the result in reverse video.

10 CLS
Clears screen.

20 PRINT "ENTER NAME:"
Prints this message on the screen.

30 $INPUT$A
Reads string input from the keyboard.

40 CLS
Clears screen.

50 XYPLOT(10,20)
Positions cursor at row 10, column 20 on the screen.

60 REV
Subsequent display will be in reverse video.

70 PRINT "HELLO", $A
Prints HELLO.

80 NRM
Cancels reverse video and returns to normal screen mode.

!RUN <RETURN>
4.1 INTRODUCTION TO FRAMEM

Introduction

FRAMEM is the Chameleon's flexible FRAMe EMulator for simulation of bit-oriented protocols. It uses all of the Chameleon BASIC commands, plus special commands and variables that control reception and transmission of bit-oriented traffic.

Using FRAMEM

Sections 4.2 and 4.3 describe the FRAMEM commands, variables and features that are available for all protocols. For this reason, you should read these sections regardless of the protocol you are using.

In addition, there are four protocol subsets of FRAMEM that make use of commands and variables that are specific to the protocol. You should also read the section that corresponds to the protocol you are using, as follows:

- Section 4.4 - FRAMEM HDLC/SDLC
- Section 4.5 - FRAMEM LAPD
- Section 4.6 - FRAMEM DMI

Remember that the Chameleon BASIC commands and variables are part of FRAMEM. These are described fully in Chapter 3.

FRAMEM I-frame

Special features of FRAMEM allow you to control the events in a frame during bit-oriented communication. FRAMEM automatically knows the correct order of a frame, including:

- Address
- Control field
- I-field
- Cyclic Redundancy Check (CRC)(ADCCP format)

The control field in an I-frame consists of the number of transmissions received \( N(r) \), a poll-final bit, the number of transmissions sent \( N(s) \), and a zero.
Figure 4.1-1 illustrates the structure of a standard ADCCP frame. The number in square brackets [ ] indicates the field width in bits.

Bits are transmitted least significant bit (LSB) first.

Figure 4.1-1: Standard ADCCP Frame

- **Addressing**: FRAMEM is highly flexible. It can handle normal or extended mode addressing, modulo 8 or modulo 128 sequence control fields, and numerous types of data fields.

- **Mnemonics**: There are built-in mnemonic tables for FRAMEM HDLC, FRAMEM SDLC and FRAMEM LAPD. You can use commands to define and save your own mnemonic tables.

- **Deviation**: FRAMEM also allows you to deviate from standard protocols during hardware/software development, and to compensate for protocol deviations in equipment supplied to you by others.
4.2 FRAMEM COMMAND INDEX

Introduction

Sections 4.2 and 4.3 describe the commands that can be used in all protocol subsets of FRAMEM (except as noted). There are additional commands and variables that are available for a specific protocol subset. These are described in the protocol-specific sections.

Remember that you can also use the Chameleon BASIC commands described in Sections 3.4 and 3.5.

Organization

In this section, the FRAMEM commands are listed by function in the following tables:

- Transmission and Reception Commands
- Mnemonic Commands
- Addressing Commands
- Miscellaneous Commands
- Transmission and Reception Variables

This functional listing provides only the command name and a brief description. For more information, refer to the page number indicated.

TABLE 4.2-1: TRANSMISSION AND RECEPTION COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORTRAN</td>
<td>4.3-2</td>
<td>S OR D</td>
<td>Transmits a frame with an abort sequence at the end.</td>
</tr>
<tr>
<td>BADTRAN</td>
<td>4.3-3</td>
<td>S OR D</td>
<td>Transmits a frame with a bad CRC.</td>
</tr>
<tr>
<td>GET</td>
<td>4.3-8</td>
<td>S OR D</td>
<td>Extracts 2 bytes from the I-field of the last frame received.</td>
</tr>
<tr>
<td>PUT</td>
<td>4.3-11</td>
<td>S OR D</td>
<td>Sets a specific byte in an I-field to a specified value.</td>
</tr>
<tr>
<td>REC</td>
<td>4.3-12</td>
<td>S OR D</td>
<td>Receives a frame from the line.</td>
</tr>
<tr>
<td>TRAN</td>
<td>4.3-27</td>
<td>S OR D</td>
<td>Transmits a frame with a good CRC.</td>
</tr>
</tbody>
</table>
### TABLE 4.2-2: MNEMONIC COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE</td>
<td>4.3-5</td>
<td>S OR D</td>
<td>Defines entries for the mnemonic table.</td>
</tr>
<tr>
<td>DEFSUB</td>
<td>4.3-6</td>
<td>S OR D</td>
<td>Defines the program line number to execute when the received frame matches a specified mnemonic.</td>
</tr>
</tbody>
</table>

### TABLE 4.2-3: ADDRESSING COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTEND</td>
<td>4.3-7</td>
<td>S OR D</td>
<td>Sets 16-bit addressing mode.</td>
</tr>
<tr>
<td>MOD</td>
<td>4.3-9</td>
<td>S OR D</td>
<td>Sets modulo 8 or 128 N(s) and N(r).</td>
</tr>
<tr>
<td>NORM</td>
<td>4.3-10</td>
<td>S OR D</td>
<td>Sets 8-bit addressing mode.</td>
</tr>
<tr>
<td>STATUS</td>
<td>4.3-25</td>
<td>S OR D</td>
<td>Displays modulo and addressing mode.</td>
</tr>
</tbody>
</table>

### TABLE 4.2-4: MISCELLANEOUS COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPRINT</td>
<td>4.3-26</td>
<td>S OR D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
TRANSMISSION AND RECEPTION VARIABLES

The variables in the table below are intended for frame transmission and reception. However, you can use them anywhere that a normal numeric variable (A-Z) can be used.

Note the use of **TX** and **RX** to denote transmission or reception of frames, respectively. Note also the use of **V(S)** and **V(R)** for **CMDR** and **FRMR** frames, respectively.

### TABLE 4.2-5: TRANSMISSION AND RECEPTION VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PAGE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>4.3-4</td>
<td>If 0, frame = good CRC. If 1, frame = bad CRC.</td>
</tr>
<tr>
<td>RXADDR</td>
<td>4.3-13</td>
<td>Received address field.</td>
</tr>
<tr>
<td>RXC/R</td>
<td>4.3-14</td>
<td>C/R bit extracted from an FRMR field.</td>
</tr>
<tr>
<td>RXDIAG</td>
<td>4.3-15</td>
<td>Last byte (WXYZ bits) of received FRMR.</td>
</tr>
<tr>
<td>RXFCTL</td>
<td>4.3-16</td>
<td>Received frame control field with poll/final bit, N(S), and N(R) removed (if applicable).</td>
</tr>
<tr>
<td>RXFRLEN</td>
<td>4.3-17</td>
<td>Length of received frame.</td>
</tr>
<tr>
<td>RXN(R)</td>
<td>4.3-18</td>
<td>Received N(R) of Supervisory frame or I-Frame.</td>
</tr>
<tr>
<td>RXN(S)</td>
<td>4.3-19</td>
<td>Received N(S) of Supervisory frame or I-Frame.</td>
</tr>
<tr>
<td>RXP/F</td>
<td>4.3-20</td>
<td>Received poll/final bit.</td>
</tr>
<tr>
<td>RXRFCTL</td>
<td>4.3-21</td>
<td>Received rejected frame control field.</td>
</tr>
<tr>
<td>RXRP/F</td>
<td>4.3-22</td>
<td>Received poll/final bit of rejected frame control field.</td>
</tr>
<tr>
<td>RXV(R)</td>
<td>4.3-23</td>
<td>Received V(R) of rejecting station.</td>
</tr>
<tr>
<td>RXV(S)</td>
<td>4.3-24</td>
<td>Received V(S) of rejecting station.</td>
</tr>
<tr>
<td>TXADDR</td>
<td>4.3-29</td>
<td>Transmitted address field.</td>
</tr>
<tr>
<td>TXC/R</td>
<td>4.3-30</td>
<td>Transmitted C/R bit of an FRMR.</td>
</tr>
<tr>
<td>TXDIAG</td>
<td>4.3-31</td>
<td>Transmitted last byte (WXYZ bits) of an FRMR field.</td>
</tr>
<tr>
<td>TXFCTL</td>
<td>4.3-32</td>
<td>Transmitted frame control field.</td>
</tr>
<tr>
<td>TXFIELD</td>
<td>4.3-33</td>
<td>Transmitted information field.</td>
</tr>
<tr>
<td>TXN(R)</td>
<td>4.3-35</td>
<td>Transmitted N(R).</td>
</tr>
<tr>
<td>TXN(S)</td>
<td>4.3-36</td>
<td>Transmitted N(S).</td>
</tr>
<tr>
<td>TXP/F</td>
<td>4.3-37</td>
<td>Transmitted poll/final bit.</td>
</tr>
<tr>
<td>TXRFCTL</td>
<td>4.3-38</td>
<td>Transmitted rejected frame control field.</td>
</tr>
<tr>
<td>TXRP/F</td>
<td>4.3-39</td>
<td>Transmitted poll/final bit of rejected frame control field.</td>
</tr>
<tr>
<td>TXV(R)</td>
<td>4.3-40</td>
<td>Transmitted V(R).</td>
</tr>
<tr>
<td>TXV(S)</td>
<td>4.3-41</td>
<td>Transmitted V(S).</td>
</tr>
</tbody>
</table>
4.3 FRAMEM COMMANDS

Introduction

This section provides a complete description of the FRAMEM commands.

If you know the name of the command you want to use, you can look up the command in this section. If you are unsure of the command you need, refer to the previous section which lists the commands by function.

The commands are described alphabetically on the following pages. The following information is provided:

- Command description
- Type of command (Statement or Direct)
- Syntax
- Command abbreviation
- Related commands (See Also)
- Example(s) of command usage
ABORTRAN

Description: The ABORTRAN command transmits a frame with an abort sequence at the end of the transmission. If the Chameleon is configured for Transparent Bisync mode, ABORTRAN acts like the TRAN command.

Type: Statement or Direct

Syntax:

```
ABORTRAN
ABORTRAN $A
```

Note: To ensure that ABORTRAN works properly, the transmitted frame must be greater than four bytes in length.

Abbreviation: AB.

See Also: BADTRAN, TRAN, Transmission variables

Example: This example transmits a frame containing the string TEST with an abort sequence.

```
10 $A = "TEST"
20 ABORTRAN $A
! RUN
```
BADTRAN

Description
The BADTRAN command transmits a frame with a bad CRC. If the Chameleon is configured for Transparent Bisync mode, BADTRAN acts like the TRAN command.

Type
Statement or Direct

Syntax
BADTRAN
BADTRAN $A

Note:
To ensure that BADTRAN works properly, the transmitted frame must be greater than four bytes in length.

Abbreviation
BA.

See Also
ABORTTRAN, TRAN, Transmission variables

Example
This example transmits a frame containing the string HELLO with a bad CRC.

10 $A = "HELLO"
20 BADTRAN $A
30 BADTRAN
40 TPRINT

(result)
T ? 00:15:16:25:30 5 48 RNR N(R) = 2 4C 4C 4F
T ? 00:15:16:27:20 2 00 IFRAME N(S) = 0 N(R) = 0

In the TPRINT display, the ? indicates a bad CRC. See TPRINT for more information.
CRC

Description
CRC is a reception variable that indicates if the received frame had a good or bad CRC. It has the following valid values:

CRC = 0 Received frame had good CRC.
CRC = 1 Received frame had bad CRC.

Type
Variable

Syntax
(See example below.)

See Also
REC, RXFRLEN

Example
This example receives a frame. If the frame has a bad CRC, it loops to receive again. When it receives a frame with a good CRC, it continues with the scenario.

10 REC
20 IF CRC = 1 GOTO 10
30 ...
DEFINE

Description The DEFINE command defines new mnemonics or redefines existing mnemonics for the mnemonic table. DEFINE is protocol specific, as described below.

Type Statement or Direct

Syntax

- **DEFINE**"name",1 = x  
  FRAMEM LAPD syntax.
- **DEFINE**"name" = x  
  All other FRAMEM protocols.

where: name is a mnemonic name of one to six characters in length and x is a numeric expression of which the lower eight bits are significant. x can be entered in decimal or hexadecimal. To enter x in hex, precede the value with an ampersand (&).

In FRAMEM LAPD, you can use the I option, which permits a specific mnemonic to have an I-field.

Abbreviation DEFI.

See Also DELETE, MLIST, MSAVE, MLOAD

Example 1 This example defines a mnemonic named NEW, with the length defined in hex.

DEFINE "NEW" = &8F
MLIST

(result) The mnemonic table is displayed with the entry:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
<th>Defsub</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>143</td>
<td>8F</td>
<td>10001111</td>
<td></td>
</tr>
</tbody>
</table>

Example 2 This example defines a FRAMEM LAPD mnemonic named NEW, with the length defined in hex, and I-field permission.

DEFINE "NEW",I = &8F
MLIST

(result) In FRAMEM LAPD, the mnemonics table is displayed with the entry:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>I-field</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>I</td>
<td>143</td>
<td>8F</td>
<td>10001111</td>
</tr>
</tbody>
</table>
DEFSUB

Description
DEFSUB defines the line number in the current program that will be executed when the received frame matches a specific mnemonic table entry.

The mnemonic must already have been defined before DEFSUB can be used.

DEFSUB defines an entry in the DEFSUB column of the mnemonic table for a specific mnemonic. The DEFSUB entry is a line number that the program jumps to when a frame is received that matches the mnemonic. It functions as if a GOSUB had been executed. DEFSUB statements remain in effect from one program to another.

Type
Statement or Direct

Syntax
DEFSUB "name" = xxxx

where: name is a mnemonic that has previously been DEFINEd, and xxxx is any valid expression that represents the line number of the program to execute if that mnemonic is received.

To remove the DEFSUB assignment, use the following syntax:
DEFSUB "name" = 0

Abbreviation
DEFS.

See Also
DEFINE, MLIST

Example
This example modifies the SABM mnemonic entry so that when the received frame is a SABM, line 100 of the current program is executed.

DEFSUB "SABM" = 100
MLIST

(result) The mnemonic table includes an entry

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
<th>Defsub</th>
</tr>
</thead>
<tbody>
<tr>
<td>SABM</td>
<td>47</td>
<td>2F</td>
<td>00101111</td>
<td>100</td>
</tr>
</tbody>
</table>
**EXTEND**

**Description**
The EXTEND command selects extended mode addressing. In EXTEND mode, 16 bits are used for the address. Changing between EXTEND and NORM (8-bit) mode automatically clears the trace buffer.

**Type**
Statement or Direct

**Syntax**
`EXTEND`

**Abbreviation**
`EXT`

**See Also**
NORM, STATUS

**Example**
This example selected extended addressing and then displays the status.

```
10 EXTEND
20 STATUS
!RUN <RETURN>
```

(result) **Modulo128/Extended addressing**
GET

**Description**
The GET command gets two bytes offset from the beginning of the received I-field. The bytes retrieved are taken as low byte, high byte. The offset is variable (see syntax).

**Type**
Statement or Direct

**Syntax**
\[ x = \text{GET} \ exp \]

**PRINT** \[ \text{GET} \ exp \]

*where:* \( \exp \) is the offset into the I-field, which can be any valid expression that evaluates to a 16-bit value. \( x \) is any valid variable. You can use the GET command with the PRINT command.

**Abbreviation**
None

**See Also**
PUT, REC, RXFRLEN

**Example 1**
This example displays bytes 1 and 2 of an I-field (second and third bytes of the I-field).

10 \( A = \text{GET} \ 1 \)
20 \( \text{PRINT} \ A \)
!RUN <RETURN>

(result) If the I-field received is: 01 02 03 04 05 06 07 08 the result 0302 is displayed.

**Example 2**
This example gets two bytes from an I-field and strips the high order byte, so that only the low order byte is extracted.

10 MOD128
20 EXTEND
30 DEFINE "MODULO" = 1
40 DEFINE "ADDR" = 1
50 REC
60 IF RXFRLEN=0 GOTO 50

70 FOR \( A = 0 \) TO RXFRLEN-(2+MODULO+ADDR)
80 \( B = \text{GET} \ A \)
90 \( \text{PRINT} \ %B \text{ AND } \&FF \)
100 NEXT \( A \)

Sets modulo 128 sequencing.
Selects extended addressing.
Defines mnemonic entry called MODULO, one bit in width.
Defines mnemonic entry called ADDR, one bit in width.
Receives a frame.
If no frame received (frame length = 0) loop to receive again.

Sets up a loop to repeat for the length of the I-field received.
Extracts two bytes beginning at byte A.
 Masks the high order byte from B.
 Increments loop counter to extract next two bytes in I-field.
MOD

Description
MOD specifies modulo 8 or modulo 128 $N_{(S)}$ and $N_{(R)}$ sequencing for frame assembly and disassembly. Changing between MOD8 and MOD128 automatically clears the trace buffer. Use the STATUS command to display the current modulo.

Type
Statement or Direct

Syntax
MOD8
MOD128

Abbreviation
None

See Also
STATUS

Example
This example sets the sequencing to modulo 8 and then displays the status.

10 MOD8
20 STATUS
!RUN  <RETURN>

(result)  Modulo8/Normal addressing
NORM

Description  The NORM command selects normal mode addressing. In normal mode, eight bits are used as the received or transmitted address. Changing between NORM and EXTEND addressing modes automatically clears the trace buffer. Use the STATUS command to display the current addressing mode.

Type  Statement or Direct

Syntax  NORM

Abbreviation  NOR.

See Also  EXTEND, STATUS

Example  This example selects normal addressing and then displays the status.

10 NORM
20 STATUS
!RUN  <RETURN>

(result)  Moduloxx/Normal addressing

where xx is either 8 or 128, depending on the previously defined modulo.
PUT

Description  The PUT command defines a specified byte in an I-field for transmission.

Type  Statement or Direct

Syntax  PUT exp1,exp2

where:  exp1 is the byte number from the start of the I-field, and exp2 is the value assigned to that byte.

Abbreviation  None

See Also  GET

Example  In this example, a value is inserted into the second byte of the I-field.

10 TXIFIELD=HEX>0000000000  Defines an I-field for transmission.

20 PUT 1,2  Defines byte 1 (the second byte) of the I-field as the value 2.

30 DISPF  Displays the most recently transmitted frame.

!RUN  <RETURN>

(result)  0002000000
**REC**

**Description**

The REC command takes the next received frame in sequence from the acquisition buffer and assigns it to zero or more string variables. A string variable can be assigned a maximum of 255 characters. When the 255-character maximum is reached, data is stored in the next string variable indicated. If there is not sufficient data in the buffer for each string variable in the command, the string variable is assigned a null string.

REC resets the RXFRLEN variable to 0, clears the DISPBUF buffer, and then transfers the frame, if any, to the trace buffer. If RXFRLEN ≠ 0, the reception variables are set depending on the protocol and what is received. For example, receiving a SABM will not affect the RXN(S) variable, which returns the received N(S).

The reception variables are listed in Section 4.2. A description of the simulation buffers is in Section 3.2.

**Type**

Statement or Direct

**Syntax**

`REC $A, $B...`

**See Also**

DEFSUB

**Example**

In this example, the scenario receives frames until a SABM is received. When a SABM is received, it transmits a UA with the appropriate poll/final bit set.

```
10 REC $A, $B, $C    Receive a frame.
20 IF RXFRLEN=0 GOTO 10   If no frame received (frame length = 0), loop to receive again.
30 IF CRC=1 GOTO 10   If bad CRC, loop to receive again.
40 IF RXFCTL≠SABM GOTO 10   If received frame is not a SABM, loop to receive again.
50 TXFCTL = UA    If SABM received, set the transmission control field to UA.
60 TXP/F = RXP/F    Set the transmission poll/final bit to match the received poll/final bit.
70 TRAN
   ! RUN <RETURN>
```

Transmit the frame.
RXADDR

Description  RXADDR equals the address field of the received frame.

Type  Variable

Syntax  (See example below.)

Abbreviation  RXA.

See Also  REC, RXFRLEN

Example  In this example, the program receives a frame and displays the address of the received frame in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 PRINT %RXADDR
RXC/R

Description
RXC/R equals the C/R bit extracted from an FRMR field.

Type
Variable

Syntax
(See example below.)

Abbreviation
RXC.

See Also
REC, RXFRLEN, RXFCTL

Example
This example receives frames until a frame reject is received, and then displays the C/R bit of the frame in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL ≠ FRMR GOTO 10
40 PRINT %RXC/R
RXDIAG

Description: RXDIAG equals the last byte of an FRMR (WXYZ bits).

Type: Variable

Syntax: (See example below.)

Abbreviation: RXD.

See Also: REC, RXFRLEN, RXFCTL

Example: This example receives frames until a frame reject is received, and then displays the last byte of the frame in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL ≠ FRMR GOTO 10
40 PRINT %RXDIAG
RXFCTL

Description
RXFCTL equals the control field of the received frame. RXFCTL removes the poll/final bit and N(S), and N(R), if applicable.

Type
Variable

Syntax
(See example below.)

Abbreviation
RXFC.

See Also
REC, RXFRLEN, RXFCTL

Example
This example receives frames until a frame with a length greater than one byte, with a good CRC is received. It then displays the control field in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFRLEN = 1 GOTO 10
40 IF CRC = 1 GOTO 10
50 PRINT %RXFCTL
60 GOTO 10
RXFRLEN

Description  RXFRLEN equals the length of the received frame.

Type        Variable

Syntax      (See example below.)

Abbreviation RXFR.

See Also    REC, RXFRLEN, RXFCTL

Example     This example loops until it receives a frame and then displays it by displaying the DISPF buffer.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 DISPF
40 GOTO 10
RXN(R)

Description  RXN(R) equals the N(R) of the received frame, if the received frame is a supervisory frame or an I-Frame.

Type Variable

Syntax  (See example below.)

Abbreviation None

See Also  REC, RXFRLEN, RXFCTL

Example  This example receives frames until an I-frame or supervisory frame is received. When the correct frame type is received, N(R) and N(S) are displayed in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL = IFRAME PRINT %RXN(R), %RXN(S)
40 IF RXFCTL = RR PRINT %RXN(R), %RXN(S)
50 IF RXFCTL = RNR PRINT %RXN(R), %RXN(S)
60 IF RXFCTL = REJ PRINT %RXN(R), %RXN(S)
70 GOTO 10
RXN(S)

Description: RXN(S) equals the N(S) of received frame, if the received frame is a supervisory frame or an I-Frame.

Type: Variable

Syntax: (See example below.)

Abbreviation: None

See Also: REC, RXFRLEN, RXFCTL

Example: This example receives frames until an I-frame or supervisory frame is received. When the correct frame type is received, N(R) and N(S) are displayed in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL = IFRAME PRINT %RXN(R), %RXN(S)
40 IF RXFCTL = RR PRINT %RXN(R), %RXN(S)
50 IF RXFCTL = RNR PRINT %RXN(R), %RXN(S)
60 IF RXFCTL = REJ PRINT %RXN(R), %RXN(S)
70 GOTO 10
## RXP/F

<table>
<thead>
<tr>
<th>Description</th>
<th>RXP/F equals the poll/final bit of the received frame.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Syntax</td>
<td>(See example below.)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>RXP.</td>
</tr>
<tr>
<td>See Also</td>
<td>REC, RXFRLEN, RXFCTL, RXRP/F</td>
</tr>
<tr>
<td>Example</td>
<td>This example sets the transmit poll/final bit to match the received poll/final bit.</td>
</tr>
</tbody>
</table>

10 REC  
20 IF RXFRLEN = 0 GOTO 10  
30 TXP/F = RXP/F
RXRFCTL

Description  RXRFCTL equals the rejected frame control field of the received frame.

Type  Variable

Syntax  (See example below.)

Abbreviation  RXRF.

See Also  REC, RXFCTL, RXFRLEN

Example  This example waits to receive a frame reject and then displays the control field of the frame.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL = FRMR PRINT %RXRFCTL
**RXRP/F**

**Description**  
RXRP/F equals the poll/final bit of a received rejected frame control field.

**Type**  
Variable

**Syntax**  
(See example below.)

**Abbreviation**  
RXRP.

**See Also**  
REC, RXFRLEN, RXRFCTL, RXP/F

**Example**  
This example waits to receive a frame reject and then displays the poll/final bit in hex.

```
10  REC
20  IF RXFRLEN = 0 GOTO 10
30  IF RXFCTL = FRMR PRINT %RXRP/F
```
RXV(R)

Description  RXV(R) equals the V(R) of the rejecting station for a rejected frame.

Type  Variable

Syntax  (See example below.)

Abbreviation  None

See Also  REC, RXFRLEN, RXRFCTL, RXRP/F

Example  This example waits to receive a frame reject and then displays the V(R) of the rejecting station in hex.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL = FRMR PRINT %RXV(R)
## RXV(S)

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>RXV(S) equals the V(S) of the rejecting station for a rejected frame.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td>(See example below.)</td>
</tr>
<tr>
<td><strong>Abbreviation</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>See Also</strong></td>
<td>REC, RXFRLEN, RXRFCTL, RXRP/F</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>This example waits to receive a frame reject and then displays the V(S) of the rejecting station in hex.</td>
</tr>
</tbody>
</table>

```plaintext
10 REC
20 IF RXFRLEN = 0 GOTO 10
30 IF RXFCTL = FRMR PRINT %RXV(S)
```
STATUS

Description  The STATUS command displays the current addressing mode and modulo.

Type  Statement or Direct

Syntax  STATUS

Abbreviation  ST.

See Also  EXTEND, NORM, MOD8, MOD128

Example  !STATUS
          (result)  Modulo8/Normal addressing
TPRINT

Description
The TPRINT command displays the contents of the trace buffer. For a general description of simulation buffers, refer to Section 3.2.

Type
Statement or Direct

Syntax
TPRINT

Abbreviation
TP.

See Also
TSAVE, TLOAD, FLUSH, TFREE

Example
10 $A = "HELLO"
20 TRAN $A
30 REC
40 TPRINT
!RUN <RETURN>
**TRAN**

**Description**

The TRAN command transmits a frame with a good CRC. It can be used to send either valid or invalid frames.

To send invalid frames, use the transmission variables to define the frame and then TRAN to transmit the frame. Using the transmission variables (Section 4.2), you can devise numerous types of frames for error testing.

Insertion of $N(S)$, $N(R)$, $V(S)$, $V(R)$, diagnostic byte, rejected frame control byte, and I-field follow ADCCP rules. In other words, if bits 1 and 0 in the Frame Control field are:

- 00 or 10 It is an I-frame.
- 11 It is a non-sequenced (unnumbered) frame
- 01 It is a supervisory frame.

A frame control byte with a value of 87 hex (CMDR/FRMR) causes $V(S)$, $V(R)$, diagnostic byte and rejected frame control byte insertion. Remember, TEST and XID cause I-field insertion.

**Note**

To maximize the speed of the Chameleon, when a TRAN is executed, control returns to the scenario before the frame is completely transmitted. If another TRAN occurs before the previous frame is transmitted, the Chameleon waits until the previous frame is finished before transmitting the second one. However, if the Chameleon is configured as a DTE and there are no clocks on the line, the second transmission causes the Chameleon to hang up. If a clock is provided the Chameleon will recover.

**Type**

Statement or Direct

**Syntax**

TRAN

TRAN $A$

**Abbreviation**

TR.

**See Also**

ABORTTRAN, BADTRAN
Example

This example sends an invalid frame by transmitting an RR frame which exceeds the required length.

10 $A = _CHR$(3) + _CHR$(1) + _CHR$(1)
20 _TRAN_ $A$

In the example above, _CHR$(3) is the address, the first _CHR$(1) is the RR, and the final _CHR$(1) is meaningless.
TXADDR

Description: TXADDR sets the value of the address field of the frame being transmitted.

Type: Variable

Syntax: (See example below.)

Abbreviation: TXA.

See Also: ABORTTRAN, BADTRAN, TRAN

Example:

10 TXADDR = 03
20 TRAN
30 DISPF
## TXC/R

**Description**  TXC/R sets the value of the command/response bit of the FRMR frame being transmitted.

**Type**  Variable

**Syntax**  (See example below.)

**Abbreviation**  TXC.

**See Also**  ABORTRAN, BADTRAN, TRAN

**Example**  

```
10 TXFCTL = FRMR  
20 TXC/R = 1  
30 TRAN  
30 DISPFL
```
TXDIAG

Description  TXDIAG sets the value of the last byte (WXYZ bits) of an FRMR field.

Type  Variable

Syntax  (See example below.)

Abbreviation  TXD.

See Also  ABORTRAN, BADTRAN, TRAN

Example  10 TXFCTL = FRMR
         20 TXDIAG = &0F
         30 TRAN
         30 DISPFL
TXFCTL

Description  TXFCTL sets the value of the frame control field of the frame being transmitted.

Type  Variable

Syntax  (See example below.)

Abbreviation  TXF.

See Also  ABORTTRAN, BADTRAN, TRAN

Example  

    10 TXFCTL = UA
    20 TRAN
    30 DISPFL
TXIFIELD

**Description**
TXIFIELD sets the contents of an I-field for a frame being transmitted. Although it is listed in the table with the transmission and reception variables, it is used like a function, rather than as a variable.

**Type**
Variable

**Syntax**
- `TXIFIELD = SA` Defines an I-field as $A$.
- `TXIFIELD + SA` Adds $A$ to the end of the existing I-field.
- `TXIFIELD = HEX > ABCD` Defines the I-field in hex.
- `TXIFIELD = ASC > ABCD` Defines the I-field in ASCII.
- `TXIFIELD = EBC > ABCD` Defines the I-field in EBCDIC.
- `TXIFIELD + HEX > 0D0A` Adds a hex string to the end of the existing I-field.
- `TXIFIELD + ASC > ABCD` Adds an ASCII string to the end of the existing I-field.
- `TXIFIELD + EBC > ABCD` Adds an EBCDIC string to the end of the existing I-field.

**Abbreviation**
TXI.

**See Also**
ABORTRAN, BADTRAN, TRAN
### Example 1

<table>
<thead>
<tr>
<th>Line</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CLEAR</td>
<td>Clears trace buffer.</td>
</tr>
<tr>
<td>10</td>
<td>TXADDR=3</td>
<td>Sets the address field to 03.</td>
</tr>
<tr>
<td>20</td>
<td>TXFCTL=0</td>
<td>Sets the frame control field to a sequenced I-frame.</td>
</tr>
<tr>
<td>30</td>
<td>TXN(S)=1</td>
<td>Sets the number of transmissions sent count to 1.</td>
</tr>
<tr>
<td>40</td>
<td>TXN(R)=2</td>
<td>Sets the number of transmissions received count to 2.</td>
</tr>
<tr>
<td>50</td>
<td>TXP/F=0</td>
<td>Sets the poll/final bit to 0.</td>
</tr>
<tr>
<td>55</td>
<td>TXIFIELD=HEX&gt;00000000000000</td>
<td>Defines the contents of the I-field in hex.</td>
</tr>
<tr>
<td>60</td>
<td>PUT 4, &amp;41</td>
<td>Sets the fifth byte of the I-field to 41 hexadecimal.</td>
</tr>
<tr>
<td>70</td>
<td>TRAN</td>
<td>Transmits the frame.</td>
</tr>
<tr>
<td>75</td>
<td>TPRINT</td>
<td>Displays trace buffer contents.</td>
</tr>
<tr>
<td>80</td>
<td>STOP</td>
<td>Terminates execution.</td>
</tr>
</tbody>
</table>

This transmits 03 42 00 00 00 00 41. Notice that unspecified bytes are set to 0.

### Example 2

Alter line 60 in the program above to read:

```
60 PUT 4, TXN(S) SHL 4  \ Sets the fifth byte of the I-field to the value of TXN(S) shifted left four bits.
```

This transmits 03 42 00 00 00 00 10.
<table>
<thead>
<tr>
<th>Description</th>
<th>TXN(R) sets the value of N(R) of the frame being transmitted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Syntax</td>
<td>(See example below.)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>None</td>
</tr>
<tr>
<td>See Also</td>
<td>ABORTTRAN, BADTRAN, TRAN</td>
</tr>
</tbody>
</table>

**Example**

```
10 TXFCTL = IFRAME
20 TXN(R) = 6
30 TXIFIELD = HEX>12345678
40 TRAN
50 DISPF
```
### TXN(S)

**Description**  
TXN(S) sets the value of N(S) of the frame being transmitted.

**Type**  
Variable

**Syntax**  
(See example below.)

**Abbreviation**  
None

**See Also**  
ABORTRAN, BADTRAN, TRAN

**Example**  
```
10 TXFCTL = IFRAME
20 TXN(S) = 6
30 TRAN
30 DISPF
```
TXP/F

Description  TXP/F sets the poll/final bit of the frame being transmitted.

Type  Variable

Syntax  (See example below.)

Abbreviation  TXP.

See Also  ABORTTRAN, BADTRAN, TRAN

Example  This example sets the transmission poll/final bit to match the poll/final bit of the received frame.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 TXP/F = RXP/F
40 TRAN
50 DISP
TXRFCTL

Description  
TXRFCTL sets the rejected frame control field of a frame being transmitted.

Type  
Variable

Syntax  
(See example below.)

Abbreviation  
TXRF

See Also  
ABORTRAN, BADTRAN, TRAN

Example  
This example sets the transmission frame control field to match the received frame control field.

10 TXRFCTL = RXFCTL
20 TRAN
30 DISP
TXRP/F

Description: TXRP/F sets the poll/final bit of a rejected frame control field for the frame being transmitted.

Type: Variable

Syntax: (See example below.)

Abbreviation: TXRP.

See Also: ABORTAN, BADTRAN, TRAN

Example:

10 TXRP/F = RXRP/F
20 TRAN
TXV(R)

Description
TXV(R) sets the value of V(R) for the frame being transmitted.

Type
Variable

Syntax
(See example below.)

Abbreviation
None

See Also
ABORTRAN, BADTRAN, TRAN

Example
10 TXV(R) = TXN(R)
20 TRAN
30 DISPF
TXV(S)

Description
TXV(S) sets the V(S) of the frame being transmitted.

Type
Variable

Syntax
(See example below.)

Abbreviation
None

See Also
ABORTRAN, BADTRAN, TRAN

Example
10 TXV(S) = TXN(S)
20 TRAN
30 DISPF
4.4 FRAMEM HDLC/SDLC

Introduction  This section describes the FRAMEM commands, parameter set-up menu and mnemonic table that are specific to FRAMEM HDLC and SDLC. These are described on the following pages.
FRAMEM HDLC AND SDLC PARAMETER SET-UP MENUS

Introduction

This section describes the parameters and valid values for the FRAMEM HDLC and SDLC menus (Figure 4.4-1). These menus enable you to configure and save parameters for running FRAMEM HDLC or SDLC simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu, refer to Section 2.3.

Figure 4.4-1: The FRAMEM (HDL/C/SDLC) Parameter Set-Up Menu

| A. Simulate          | DCE       |
| B. Data Encoding    | NRZ       |
| C. Coupler          | HDLC/SDLC |
| D. Bit Rate         | 56000     |

SELECT >

INPUT: SELECT a letter or type an option key shown below

Z = run FRAMEM, ? = Help, S = Save parameters, ESC = Main Menu

A. Simulate
   - DCE
   - DTE

B. Data Encoding
   - NRZ
   - NRZI

C. Coupler
   - HDLC/SDLC
   - XPARENT BISYNC (See additional parameters on page 4.4-4.)

D. Bit Rate
   - Decimal bit rate range: 50-64000
If you are unfamiliar with these parameters, use the following guidelines for assistance in selecting the correct settings.

**Simulate:**

**DCE/DTE**

The DCE furnishes the clock which determines the data rate. It also determines on which connector pins the data is physically sent and received. For Simulation, one device is designated as the DCE; the other device is designated as a DTE.

If the Unit Under Test (UUT) is supplying the clock, it is the DCE. In this case, the Chameleon must be configured as a DTE. If the UUT is not supplying the clock, it is designated as a DTE. The Chameleon must then be configured as the DCE.

**Data Encoding:**

**NRZ/NRZI**

This parameter determines if the data is transmitted in NRZ (Non-Return to Zero) or NRZI (Non-Return to Zero Inverted) format. This is a function of the relative voltage on the data line, representing the 'MARK' and 'SPACE' levels. This parameter determines whether a 'MARK' is represented by a high or low level.

Configure the Chameleon to use the same data encoding format as the (UUT).

**Coupler:**

**HDLC/SDLC or XPARENT BSC**

This parameter enables you to select the HDLC/SDLC coupler or the Transparent BSC coupler. The HDLC/SDLC coupler is for X.25/SNA applications in FRAMEM. The Transparent BSC coupler supports the X.25/ECMA protocol, which is used in some European countries.

Select the coupler option that is appropriate for the protocol you are testing. If you select the Transparent BSC option, 10 additional parameters are displayed in the set-up menu, as described on page 4.4-4.

**Bit Rate**

The bit rate determines the speed that the Chameleon 32 will transmit data. If the Chameleon is configured as the DTE, the bit rate is automatically set to RECEIVE. This means that the Chameleon will send data at the same rate as the data it receives, thus matching the speed set by the UUT.

If the Chameleon is configured as the DCE, it is responsible for setting the bit rate. The valid range is 50 - 64000 bits per second.
Xparent Bisync Menu

If you choose the Xparent Bisync value for the C. - Coupler parameter, 10 additional parameters (E - N) are displayed as shown in the figure below.

This is the Transparent BSC menu that configures the parameters necessary for simulating the X.25/ECMA protocol.

---

**FRAMEM PARAMETER SET-UP MENU**

| A. Simulate | :DCE | G. Sync 2 | :32 | M. ENQ | :23 |
| B. Data Encoding | :NRZ | H. SOH | :01 | N. ETB | :26 |
| C. Coupler | :XPARENT BISYNC | I. STX | :02 |
| D. Bit Rate | :56000 | J. ETX | :03 |
| E. Code | :EBCDIC | K. DLE | :10 |
| F. Sync 1 | :32 | L. EOT | :37 |

SELECT: >

CHOICES:

INPUT: SELECT a letter (A - N)
or type an option key shown below

Z = Run Framem.  ? = Help.  S = Save parameters.  ESC = Main Menu

---

Figure 4.4-2: Xparent Bisync Parameters

The additional Xparent Bisync parameters (E - N) are described below.

E. Code
- EBCDIC
- ASCII/ NO PARITY
- ASCII/ EVEN PARITY
- ASCII/ ODD PARITY

F. Sync 1
G. Sync 2
H. SOH (Start of Header)
I. STX (Start of Text)
J. ETX (End of Text)
K. DLE (Data Link Escape)
L. EOT (End of Transmission)
M. ENQ (Enquiry)
N. ETB (End of Transmission Block)

2-digit hex number
FRAMES HDLC/SDLC MNEMONIC TABLE

Introduction

The FRAMES HDLC/SDLC mnemonic table has five columns:

- Mnemonic name (1-6 characters)
- Decimal value
- Hexadecimal value
- Binary value
- DEFSUB - If this type of frame is received, program control jumps to the program line number specified in this column and executes the subroutine. Refer to the FRAMES DEFSUB command for more information.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>BINARY</th>
<th>DEFSUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAME</td>
<td>0</td>
<td>00</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>SNRME</td>
<td>207</td>
<td>CF</td>
<td>11001111</td>
<td></td>
</tr>
<tr>
<td>SARME</td>
<td>79</td>
<td>4F</td>
<td>01001111</td>
<td></td>
</tr>
<tr>
<td>SABME</td>
<td>111</td>
<td>6F</td>
<td>01101111</td>
<td></td>
</tr>
<tr>
<td>SREJ</td>
<td>13</td>
<td>0D</td>
<td>00001101</td>
<td></td>
</tr>
<tr>
<td>SNRM</td>
<td>131</td>
<td>83</td>
<td>10000011</td>
<td></td>
</tr>
<tr>
<td>SARM</td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
<td></td>
</tr>
<tr>
<td>SABM</td>
<td>47</td>
<td>2F</td>
<td>00101111</td>
<td></td>
</tr>
<tr>
<td>DISC</td>
<td>67</td>
<td>43</td>
<td>01000011</td>
<td></td>
</tr>
<tr>
<td>RSET</td>
<td>143</td>
<td>8F</td>
<td>10001111</td>
<td></td>
</tr>
<tr>
<td>FRMR</td>
<td>135</td>
<td>87</td>
<td>10000111</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>227</td>
<td>E3</td>
<td>11100011</td>
<td></td>
</tr>
<tr>
<td>CMDR</td>
<td>135</td>
<td>87</td>
<td>10000111</td>
<td></td>
</tr>
<tr>
<td>RNR</td>
<td>5</td>
<td>05</td>
<td>00000101</td>
<td></td>
</tr>
<tr>
<td>REJ</td>
<td>9</td>
<td>09</td>
<td>00001001</td>
<td></td>
</tr>
<tr>
<td>SIM</td>
<td>7</td>
<td>07</td>
<td>00000111</td>
<td></td>
</tr>
<tr>
<td>XID</td>
<td>175</td>
<td>AF</td>
<td>10101111</td>
<td></td>
</tr>
<tr>
<td>RIM</td>
<td>7</td>
<td>07</td>
<td>00000111</td>
<td></td>
</tr>
<tr>
<td>NSI</td>
<td>3</td>
<td>03</td>
<td>00000011</td>
<td></td>
</tr>
<tr>
<td>RQI</td>
<td>7</td>
<td>07</td>
<td>00000111</td>
<td></td>
</tr>
<tr>
<td>ROL</td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
<td></td>
</tr>
<tr>
<td>NSP</td>
<td>35</td>
<td>23</td>
<td>00100011</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
<td></td>
</tr>
<tr>
<td>UI</td>
<td>3</td>
<td>03</td>
<td>00000011</td>
<td></td>
</tr>
<tr>
<td>UP</td>
<td>35</td>
<td>23</td>
<td>00100011</td>
<td></td>
</tr>
<tr>
<td>DM</td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
<td></td>
</tr>
<tr>
<td>UA</td>
<td>99</td>
<td>63</td>
<td>01100011</td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td>67</td>
<td>43</td>
<td>01000011</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4-3: FRAMES HDLC/SDLC Mnemonic Table
FRAMEM HDLC/SDLC COMMANDS

There are no FRAMEM commands that are specific to HDLC and SDLC; however, the TPRINT command, which displays the contents of the trace buffer, has a protocol-specific display format. The trace buffer display for FRAMEM HDLC and FRAMEM SDLC is shown on the following page.

Remember that for FRAMEM HDLC/SDLC simulation, you can use the FRAMEM commands listed in Section 4.3 and the Chameleon BASIC commands listed in Section 3.5.
TPRINT

Description
The TPRINT command displays the contents of the trace buffer, which contains the received and transmitted traffic.

The format of the trace buffer display is protocol-specific as shown in the examples below. For more information about the use of the trace buffer in simulation, refer to Section 3.2.

Type
Statement or Direct

Syntax
TPRINT

Abbreviation
TP.

See Also
TSAVE, TLOAD, REC

Example 1
This is an example of a FRAMEM HDLC/SDLC trace buffer display.

R 14:26:06:77:90 4 01 IFRAME N(S) =2 N(R) =3 00 00
R 14:26:08:77:90 2 01 P SABM
T 14:26:09:17:90 5 43 FRMR 97 F8 07
FRAMEM HDLC/SDLC SAMPLE PROGRAMS

The following sample FRAMEM HDLC/SDLC programs illustrate important concepts for developing applications, but are not valid tests for any particular application.

Program FHDLC1

This program waits for a SABM frame to be received on any channel. After receiving one, it sends a UA frame on channel 03.

5 CLEAR
10 REM WAIT FOR SABM
20 REC
30 IF RXFRLEN=0 GOTO 20
40 IF RXFCTL#SABM GOTO 20
50 TXFCTL=UA
60 TXADDR=03
70 TRAN
80 STOP

5 CLEAR
Clears trace buffer.

10 REM WAIT FOR SABM
Remark, not executed.

20 REC
Receives a frame, if one is available.

30 IF RXFRLEN=0 GOTO 20
If no frame is received, RXFRLEN equals 0, loops back to line 20, and tries again.

40 IF RXFCTL#SABM GOTO 20
If the received frame is not a SABM, loops back to line 20, and tries again.

50 TXFCTL=UA
A SABM was received. Prepares to transmit a UA by setting the transmit frame control field to UA.

60 TXADDR=03
Sets transmit address to 03.

70 TRAN
Transmits the frame.

80 STOP
Terminates execution.
Program FHDLC2 This program receives a number of frames, and displays the frame control field.

10 REC
20 IF RXFRLEN = 0 GOTO 10
30 PRINT "FRAME CONTROL = " , RXFCTL
40 STOP

Receives a frame.
If no frame received, loops to receive again.
Prints the control field in hex.
Terminates execution.

Program FHDLC3 This program waits to receive a SABM or UA and then displays the contents of the DISPF buffer and the trace buffer.

10 DEFSUB "SABM"=100
20 DEFSUB "UA"=100
25 DEFSUB "SARM"=300
30 A = 0
35 REC
40 IF (RXFRLEN=0) OR (A=0) GOTO 30
50 PRINT "NO SABM OR UA"
60 GOTO 30
70 DISPF
80 TPRINT
90 CLEAR
100 A = 1
110 RETURN
300 PRINT "BYE"
320 STOP

Subroutine if SABM received.
Subroutine if UA received.
Subroutine if SARM received.
Sets flag to 0.
Receives a frame.
If nothing was received, or if flag still equals 0, goes to 30.
Displays error message.
Tries SABM/UA subroutine again.
Displays last received frame.
Displays contents of trace.
Clears trace buffer.
Sets flag to one.
End of subroutine.
SARM subroutine.
Terminates execution.
This program does the following:

- Establishes a link
- Sends and receives a user-specified number of I-frames within a FOR loop
- Disconnects the link

Flow control N(R) and N(S) are monitored. If an error occurs, control exits from the loop, prints an appropriate message, and disconnects the link. This example should be run with the Chameleon simulating the DTE side.

```
1 CLS                   Clear screen.
2 INPUT "ENTER # OF SECONDS OF I-FRAME RECEPTION DELAY" B
   Prompts for timer value and stores result in variable B.
10 INPUT "ENTER # OF I-FRAMES TO BE SENT" C
   Prompts for I-frames to transmit and stores result in variable C.
20 CLEAR                 Clears trace buffer.
30 TXADDR = 01           Sets transmission address.
40 TXN(S) = 0            Sets N(S) for transmitted frame.
50 TXN(R) = 0            Sets N(R) for transmitted frame.
70 TXFCTL = SABM         Sets control field for transmitted frame.
90 TRAN                  Transmits frame.
130 REC                  Receives a frame.
150 IF (RXFRLEN=0) OR (RXFCTL#UA) GOTO 130
   If no frame received, or received other than UA, loop to receive again.
155 IF C = 0 GOTO 920    If all I-frames have been received, leaves loop and sends DISC.
160 C = C - 1            Decrements I-frame counter.
170 FOR A = 0 TO C       Loop index becomes number of I-frames.
180 TXADDR = 01          Sets transmission address.
190 TXFCTL=IFRAME        Sets control field to I-frame.
200 TXIFIELD=HEX>000000000000000000000000000000000
   Sets I-field value.
```
210 TRAN
250 TXN(S)=TXN(S)+1
260 REC

Transmits frame.
Increments number sent count N(S).
Receives a frame.

270 IF (RXFRLEN=0) OR (RXFCTL=RR) GOTO 260

If a frame is not received, or the frame received is not an RR, loops to receive again.

280 IF RXN(R)$TXN(S) PRINT "RXN(R) IS OUT OF SEQUENCE"

Checks flow control.

290 TIM2 = B
300 REC

Sets timeout counter.
Receives a frame.

305 IF TIM2=0 GOTO 900

If timer expires, goes to 900.

310 IF (RXFRLEN=0) OR (RXFCTL$IFRAME) GOTO 300

If a frame is not received, or the frame received is not an I-frame, loops to receive again.

330 IF RXN(S)$TXN(R) PRINT "RXN(S) IS OUT OF SEQUENCE" GOTO 920

340 IF TXN(S) = RXN(R) GOTO 360

350 PRINT "RXN(R) IS OUT OF SEQUENCE": GOTO 920
360 TXADDR=03
370 TXFCTL=RR
380 TXN(R)=TXN(R)+1
390 TRAN

Sets transmit address to 03.
Sets control field to RR.
Increments received frames count.
Transmits the frame.

900 NEXT A
920 TXADDR=01
930 TXFCTL=DISC
940 TRAN
950 REC

Increases loop index, return to 170.
Builds and transmits DISC frame.
Sets frame control field.
Transmits the frame.
Receives frame.

960 IF (RXFRLEN=0) OR (RXFCTL=UA) GOTO 950

If no frame received, or frame received is not an UA, loops to receive again.

970 STOP

Terminates program.
Program HDLC5

This program is valid only in FRAMEM HDLC. It assembles and transmits every common type of HDLC/X.25 packet. The packets are sent on the primary and secondary addresses, as appropriate, both with and without the poll bit set.

10 IF B=1 GOTO 60  
   B is a flag.
20 GOSUB 890  
   Defines strings as packets.
30 GOSUB 460  
   DCE or DTE? checks addresses.
40 B=1  
   Sets flag.
50 COUPLER "DEFAULT"  
   Calls coupler set-up routine.
60 INPUT "HOW MANY TIMES THRU LOOP" Y  
   Gets loop index.
70 FOR Z=1 TO Y  
   Begins loop.
80 CLS  
   Clears screen.
90 PRINT "TESTING SUPERVISORY FRAMES"  
100 TXFCTL=RR  
   Sets Frame Control Field to RR
110 GOSUB 380  
   Transmits primary/secondary.
120 TXFCTL=RNR  
   Sets Frame Control Field to RNR.
130 GOSUB 380  
   Transmits primary/secondary.
140 TXFCTL=REJ  
   Sets Frame Control Field to REJ.
150 GOSUB 380  
   Transmits primary/secondary.
160 PRINT "TESTING UN-NUMBERED COMMANDS"  
170 TXADDR=P  
   Sets address to primary.
180 TXFCTL=SARM  
   Sets control field to SARM.
190 GOSUB 410  
   Transmits primary only.
200 TXFCTL=SABM  
   Sets control field to SABM.
210 GOSUB 410  
   Transmits primary only.
220 TXFCTL=DISC  
   Sets control field to DISC.
230 GOSUB 410  
   Transmits primary only.
240 PRINT "TESTING UN-NUMBERED RESPONSES"  
250 TXADDR=S  
   Sets address to secondary.
260 TXFCTL=DM  Sets control field to DM.
270 GOSUB 410  Transmits secondary only.
280 TXFCTL=FRMR Sets control field to FRMR.
290 GOSUB 410  Transmits secondary only.
300 TXFCTL=UA  Sets control Field to UA.
310 GOSUB 410  Transmits secondary only.
320 PRINT "TESTING I-FRAME" Displays message.
330 TXADDR=P  Sets address to primary.
340 TXFCTL=I-FRAME Sets control field to I-frame.
350 GOSUB 580  Transmits I-frames.
360 NEXT Z  Increments loop index.
370 STOP  Terminates execution.
380 TXADDR=P  Sets address to primary.
390 GOSUB 410  Transmit routine.
400 TXADDR=S  Sets address to secondary.
410 TXP/F=0  Sets poll bit to zero.
420 TRAN  Transmits frame without poll.
430 TXP/F=1  Sets poll bit to one.
440 TRAN  Transmits frame with poll.
450 RETURN  Ends subroutine 410.
460 CLS  Clears screen.
470 XYPLOT(10,20) Positions cursor.
480 PRINT "ARE YOU A DCE? (Y/N)" Displays question.
490 $INPUT $X  Assigns user input to $X.
500 IF LEN($X)=1 GOTO 460  If user input invalid, try again.
510 IF $X="Y" GOTO 540  If DCE, goes to 540.
520 IF $X="N" GOTO 560  If DTE, goes to 560.
530 GOTO 460  If invalid input, try again.
Assigns address variables.
Ends subroutine 540.
Assigns address variables.
Ends subroutine 560.
Line 580 - 880 assign and transmit fields using string variables.

540 P=3:S=1
550 RETURN
560 P=1:S=3
570 RETURN
580 TXIFIELD=$A

590 TRAN
600 TXIFIELD=$B
610 TRAN
620 TXIFIELD=$C
630 TRAN
640 TXIFIELD=$D
650 TRAN
660 TXIFIELD=$E
670 TRAN
680 TXIFIELD=$F
690 TRAN
700 TXIFIELD=$G
710 TRAN
720 TXIFIELD=$H
730 TRAN
740 TXIFIELD=$I
750 TRAN
760 TXIFIELD=$J
770 TRAN
780 TXIFIELD=$K
790 TRAN
800 TXIFIELD=$L
810 TRAN
820 TXIFIELD=$M
830 TRAN
840 TXIFIELD=$N
850 TRAN
860 TXIFIELD=$O
870 TRAN
880 RETURN

End of subroutine 580.

890 $A=HEX>1001 00
900 $A=SA++ABCD$EFGHIJKLMNOPQRSTUVWXYZ0123456789*
910 $B=HEX>1001 01
920 $C=HEX>1001 05
930 $D=HEX>1001 09
940 $E=HEX>1001 23
950 $F=HEX>1001 27
960 $G=HEX>1001 08 07 543210000
970 $H=HEX>1001 0F
980 $I=HEX>1001 13 00
990 $J=HEX>1001 17
1000 $K=HEX>1001 1B 00
1010 $L=HEX>1001 1F
1020 $M=HEX>1001 F8 00
1030 $N=HEX>1001 FF
1040 $O=HEX>1001 F1 12345678
1050 RETURN

$A is a data packet.
$B is a RR.
$C is a RNR.
$D is a REJECT.
$E is an Interrupt.
$F is an Interrupt Confirmation.
$G is a Call.
$H is a Call Confirmation.
$I is a Clear.
$J is a Clear Confirmation.
$K is a Reset.
$L is a Reset Confirmation.
$M is a Restart.
$N is a Restart Confirmation.
$O is a Diagnostic Packet.
End of subroutine 890.
Program HDLC6

This program is valid only for FRAMEM HDLC. It demonstrates the use of the RXRFCTL, RXDIAG, RXRP/F, RXV(S) AND RXV(R) variables, and the DELETE and DEFSUB commands. It should be run on the DTE side in conjunction with Program HDLC7 on the following page.

5 CLS
10 FLUSH
20 CLEAR
30 DELETE "SARM"
40 DELETE "ROL"
50 DELETE "FRMR"
60 DEFSUB "CMDR"=2000
70 DEFSUB "SABM"=1000
75 TXP/F=0
80 REC
90 GOTO 80
1000 TXFCTL=DM
1005 DISPF
1010 TXADDR=03
1020 TRAN
1030 RETURN
2000 DISPF
2020 ?"RXRFCTL=","RXRFCTL","RXDIAG=","RXDIAG","RXRP/F=","RXRP/F"
2025 ? "RXV(S)=","RXV(S)","RXV(R)=","RXV(R)"

Clears screen
Clears acquisition buffer.
Clears trace buffer.
deletes SARM mnemonic.
deletes ROL mnemonic.
deletes FRMR mnemonic.
jumps to line 2000 if CMDR received.
jumps to line 1000 if SABM received.
sets poll final bit to 0.
receives a frame.
loops until a CMDR or SABM is received, and then executes subroutine defined in DEFSUB commands in lines 60 and 70.
sets transmit control field to DM.
displays last frame received.
sets transmit address to 03.
transmits frame.
returns from subroutine.
displays last frame transmitted.
displays values in hex.
displays values of RXV(S) and RXV(R).
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2030</td>
<td>TXP/F=1</td>
<td>Sets poll final bit.</td>
</tr>
<tr>
<td>2040</td>
<td>IF RXRP/F=0 GOTO 2070</td>
<td>If received frame reject with poll/final bit not set, jump to line 2070.</td>
</tr>
<tr>
<td>2050</td>
<td>TPRINT</td>
<td>Displays trace buffer contents.</td>
</tr>
<tr>
<td>2060</td>
<td>STOP</td>
<td>Terminates execution.</td>
</tr>
<tr>
<td>2070</td>
<td>TRAN</td>
<td>Transmits frame with poll bit.</td>
</tr>
<tr>
<td>2080</td>
<td>DISPF</td>
<td>Displays last frame transmitted.</td>
</tr>
<tr>
<td>2090</td>
<td>RETURN</td>
<td>Returns from subroutine.</td>
</tr>
</tbody>
</table>
Program HDLC7

This program demonstrates the use of the TXRFCTL, TXDIAG, TXV(S), TXV(R) and TXRP/F variables, which are used to transmit a CMDR or FRMR. This scenario should be run on the DCE side in conjunction with Program HDLC6 on the preceding page.

1   CLS                   Clears screen.
2   REM DELETE MNEMONICS WITH SAME VALUE AS ONE YOU ARE
3   REM LOOKING FOR. ROL AND SARM EQUAL &OF AS DOES DM.
4   DELETE "ROL"            Deletes mnemonic from table.
5   DELETE "SARM"            Deletes mnemonic from table.
6   REM FRMR ALSO HAS THE SAME CODE AS CMDR.
7   DELETE "FRMR"            Deletes mnemonic from table.
10  DEF SUB "DM"=500          if DM received, jumps to line 500 as a subroutine call.
20  TXP/F=0                 Sets poll final bit to 0.
30  FLUSH                   Clears receive frame buffer.
40  CLEAR                   Clears trace buffer.
45  TXADDR=03               Sets transmit address to 03.
50  TXFCTL=SABM             Sets frame control field to SABM.
70  TXV(S)=0                Sets V(S) equal to 0.
80  TXV(R)=0                Sets V(R) equal to 0.
90  TRAN                    Transmits frame.
95  DISPF                   Displays last frame received.
100 REC                     Receives frame.
110 GOTO 100                Loops until frame is received.
500 TXRFCTL=RXFCTL          DM subroutine. Assigns transmit control field (DM) to rejected frame field of CMDR to transmit.
510 TXRP/F=RXP/F            Assigns received poll final bit to transmit poll final bit.
520 TXFCTL=CMDR  Transmit sframe control is equal to CMDR.
525 DISP  Displays last frame transmitted.
530 ?“RECEIVED DM.  SENDING CMDR WITH ‘W,X,Y AND Z’ BITS SET”.
540 TXDIAG=8F  Transmits diagnostic field of F hex.
560 TRAN  Transmits CMDR frame.
570 DISP  Displays last frame transmitted.
580 IF TXRP/F=1 GOTO 610  If received DM has poll bit set, jumps to line 610.
590 TXP/F=1  Sets poll bit to 1.
600 RETURN  Ends subroutine.
610 TPRINT  Displays trace buffer.
620 STOP  Ends program.

The figure below shows the sequence that occurs in programs HDLC6 and HDLC7.

![Diagram](image-url)

DCE (HDLC7)  
\[\text{SABM} \rightarrow \text{DM} \rightarrow \text{DTE (HDLC6)}\]

CMDR W, X, Y, Z
4.5 FRAMEM LAPD

Introduction
This section describes the commands, variables, menus, and mnemonics table that are specific to FRAMEM LAPD.

LAPD Frame
The structure of a LAPD frame is slightly different from a standard ADDCP frame. Specifically, the LAPD address field has several added bits, as shown in Figure 4.5-1. The number in parentheses ( ) indicates the width of the field in bits.

<table>
<thead>
<tr>
<th>Flag (8 bits)</th>
<th>Address (16 bits)</th>
<th>Control Field (8 or 16 bits)</th>
<th>Information (variable length)</th>
<th>Frame Check Sequence (16 bits)</th>
<th>Flag (8 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7E hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7E hex</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SAPI</th>
<th>C/R</th>
<th>EA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEI</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MODULO 8

<table>
<thead>
<tr>
<th>I-frame format</th>
<th>S-frame format</th>
<th>U-frame format</th>
</tr>
</thead>
<tbody>
<tr>
<td>N(R) P N(S) 0</td>
<td>N(R) P/ F S S 0 1</td>
<td>M M M P/ F M M 1 1</td>
</tr>
</tbody>
</table>

MODULO 128

<table>
<thead>
<tr>
<th>I-frame format</th>
<th>S-frame format</th>
<th>U-frame format</th>
</tr>
</thead>
<tbody>
<tr>
<td>N(S) 0</td>
<td>N(R) P</td>
<td>M M M P/ F M M 1 1</td>
</tr>
</tbody>
</table>

SAPI: Service Access Point Identifier
- 0 = Call control procedures
- 16 = Packet communication procedures
- 32-47 = Reserved for national use
- 63 = Management procedures
- All others = Reserved for future standard

TEI: Terminal Endpoint Identifier
- 0-63 = Non-automatic TEI assignment equipment
- 64-126 = Automatic TEI assignment equipment
- 127 = Broadcast (group) TEI

C/R: Command/Response bit
- From user side: 1 = response; 0 = command
- From network side: 1 = command, 0 = response

EA: Extended Address
- 0 = Not final address octet
- 1 = Final octet of address field

Figure 4.5-1: LAPD Frame Format
CAPABILITIES OF FRAMEM LAPD

With the FRAMEM LAPD package, you can specify and transmit frames with any combination of the following errors:

- Introduce a frame that is too short, by setting a variable equal to a single hexadecimal character, instead of the normal hexadecimal pair.
- Send a frame or an information field that is too long, by setting the I-field to a value greater than the user-defined N1 (maximum frame length) value.
- Transmit an unknown frame by defining a mnemonic that will not be recognized by the device or the network under test.
- Transmit an invalid first order bit by setting the bit if it should be clear, or clearing the bit if it should be set.
- Send an invalid N(R) or N(S) by setting N(R) to a greater number than N(S), or defining both to be greater than the window.
- Introduce an incomplete frame by sending an I-frame with an empty I-field.
- Send a valid or invalid FCS using the TRAN and BADTRAN commands.
- Specify frame type, content, and component parts of both the control and address fields using standard LAPD or user-defined mnemonics.
- Define control fields by bit pattern and the associated mnemonic identifiers, and specify data content as a series of hexadecimal bytes for frames with I-fields.
FRAMESM LAPD PARAMETER SET-UP MENU

Introduction

This section describes the parameters and valid values for the FRAMESM LAPD menu (Figure 4.5-2). This menu enables you to configure and save parameters for running FRAMESM LAPD simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu (option S), refer to Section 2.3.

Figure 4.5-2: The FRAMESM LAPD Parameter Set-Up Menu

- A. Simulate: DCE
  - DTE
- B. Data Encoding: NRZ
  - NRZI
- C. Interframe Fill: FF hex
  - 7E hex
- D. Bit Rate: 56000

Decimal bit rate range: 50 - 64000
If you are unfamiliar with these parameters, use the following guidelines for assistance in selecting the correct settings.

**Simulate:**

**DCE/DTE**

The DCE furnishes the clock which determines the data rate. It also determines on which connector pins the data is physically sent and received. For Simulation, one device is designated as the DCE; the other device as a DTE.

If the Unit Under Test (UUT) is supplying the clock, it is the DCE. In this case, the Chameleon must be configured as a DTE. If the UUT is not supplying the clock, it is designated as a DTE. The Chameleon is then configured as the DCE.

**Data Encoding:**

**NRZ/NRZI**

This parameter determines if the data is transmitted in NRZ (Non-Return to Zero) or NRZI (Non-Return to Zero Inverted) format. This is a function of the relative voltage on the data line, representing the 'MARK' and 'SPACE' levels. This parameter determines whether a 'MARK' is represented by a high or low level. Configure the Chameleon to use the same data encoding format as the (UUT).

**Interframe Fill:**

**7E/FF**

The Interframe fill determines what the Chameleon sends to the UUT when the data line is idle. The original CCITT recommendation specified an all 1’s (FF) condition to indicate the line is idle, and 7E to indicate the line is busy.

Many terminal devices require an interframe fill value of FF, since an interframe fill of 7E can be mistaken as the beginning and ending flags of an endless stream of zero-length frames. If the interframe fill is mistaken in this way, the terminal becomes so busy that it appears to be locked up.

Many switch devices assume that the physical link is disconnected or faulty unless there is a continuous stream of bits, consisting of data or hex 7E flags, on the line.

**Bit Rate**

The bit rate determines the speed that the Chameleon 32 will transmit data. If the Chameleon is configured as the DTE, the bit rate is automatically set to RECEIVE. This means that the Chameleon will send data at the same rate as the data it receives, thus matching the speed set by the UUT.

If the Chameleon is configured as the DCE, it is responsible for setting the bit rate. The valid range is 50 - 64000 bits per second.
The FRAMEM LAPD mnemonic table has five fields:

- Mnemonic name (1 - 6 characters)
- I-field permission
- Decimal value
- Hexadecimal value
- Binary value

The default FRAMEM LAPD mnemonic table is shown in the table below. The I-field table in the column indicates whether the mnemonic can have an I-field. If an I-field is permitted, the letter I appears in the second column next to the mnemonic name.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>I-FIELD</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAME</td>
<td></td>
<td>0</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>SNRME</td>
<td></td>
<td>207</td>
<td>CF</td>
<td>11001111</td>
</tr>
<tr>
<td>SARME</td>
<td></td>
<td>79</td>
<td>4F</td>
<td>01001111</td>
</tr>
<tr>
<td>SABME</td>
<td></td>
<td>111</td>
<td>6F</td>
<td>01101111</td>
</tr>
<tr>
<td>SREJ</td>
<td></td>
<td>13</td>
<td>0D</td>
<td>00001101</td>
</tr>
<tr>
<td>SNRM</td>
<td></td>
<td>131</td>
<td>83</td>
<td>10000011</td>
</tr>
<tr>
<td>SARM</td>
<td></td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
</tr>
<tr>
<td>SABM</td>
<td></td>
<td>47</td>
<td>2F</td>
<td>00101111</td>
</tr>
<tr>
<td>DISC</td>
<td></td>
<td>67</td>
<td>43</td>
<td>01000011</td>
</tr>
<tr>
<td>RSET</td>
<td></td>
<td>143</td>
<td>8F</td>
<td>10001111</td>
</tr>
<tr>
<td>FRMR</td>
<td></td>
<td>135</td>
<td>87</td>
<td>10000111</td>
</tr>
<tr>
<td>TEST</td>
<td></td>
<td>227</td>
<td>E3</td>
<td>11100011</td>
</tr>
<tr>
<td>CMDR</td>
<td></td>
<td>135</td>
<td>87</td>
<td>10000111</td>
</tr>
<tr>
<td>RNR</td>
<td></td>
<td>5</td>
<td>05</td>
<td>00000101</td>
</tr>
<tr>
<td>REJ</td>
<td></td>
<td>9</td>
<td>09</td>
<td>00001101</td>
</tr>
<tr>
<td>SIM</td>
<td></td>
<td>7</td>
<td>07</td>
<td>00000111</td>
</tr>
<tr>
<td>XID</td>
<td></td>
<td>175</td>
<td>AF</td>
<td>10101111</td>
</tr>
<tr>
<td>RIM</td>
<td></td>
<td>7</td>
<td>07</td>
<td>00000111</td>
</tr>
<tr>
<td>NSI</td>
<td></td>
<td>3</td>
<td>03</td>
<td>00000111</td>
</tr>
<tr>
<td>RQI</td>
<td></td>
<td>7</td>
<td>07</td>
<td>00000111</td>
</tr>
<tr>
<td>ROL</td>
<td></td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
</tr>
<tr>
<td>NSP</td>
<td></td>
<td>35</td>
<td>23</td>
<td>00100011</td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td>1</td>
<td>01</td>
<td>00000101</td>
</tr>
<tr>
<td>UI</td>
<td></td>
<td>3</td>
<td>03</td>
<td>00000111</td>
</tr>
<tr>
<td>UP</td>
<td></td>
<td>35</td>
<td>23</td>
<td>00100011</td>
</tr>
<tr>
<td>DM</td>
<td></td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
</tr>
<tr>
<td>UA</td>
<td></td>
<td>99</td>
<td>63</td>
<td>01100011</td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td>67</td>
<td>43</td>
<td>01000011</td>
</tr>
</tbody>
</table>

Figure 4.5-3: FRAMEM LAPD Mnemonic Table
FRAMEM LAPD COMMAND INDEX

The commands that are available only in FRAMEM LAPD, or have syntax that differs from other BASIC subsets are described in this section. This page lists the FRAMEM LAPD commands and variables in tables by function. It also indicates whether the command is a Statement or Direct command. For more information, refer to the page number indicated.

Remember that you can also use the FRAMEM commands listed in Section 4.3 the Chameleon BASIC commands listed in Section 3.5.

### TABLE 4.5-4: FRAMEM LAPD TRANSMISSION AND RECEPTION VARIABLES

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXCR</td>
<td>4.5-9</td>
<td>Variable</td>
<td>Indicates if received C/R bit is set.</td>
</tr>
<tr>
<td>RXSAPI</td>
<td>4.5-10</td>
<td>Variable</td>
<td>Received Service Access Point Identifier.</td>
</tr>
<tr>
<td>RXTEI</td>
<td>4.5-11</td>
<td>Variable</td>
<td>Received Terminal Endpoint Identifier.</td>
</tr>
<tr>
<td>TXCR</td>
<td>4.5-13</td>
<td>Variable</td>
<td>Indicates if transmitted C/R bit is set.</td>
</tr>
<tr>
<td>TXSAPI</td>
<td>4.5-14</td>
<td>Variable</td>
<td>Transmitted Service Access Point Identifier.</td>
</tr>
<tr>
<td>TXTEI</td>
<td>4.5-15</td>
<td>Variable</td>
<td>Transmitted Terminal Endpoint Identifier.</td>
</tr>
</tbody>
</table>

### TABLE 4.5-5: FRAMEM LAPD MISCELLANEOUS COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE</td>
<td>4.5-7</td>
<td>S or D</td>
<td>Defines a new mnemonic, or redefines an existing mnemonic for the table</td>
</tr>
<tr>
<td>FILL</td>
<td>4.5-8</td>
<td>S or D</td>
<td>Defines the interframe fill as either 7E or FF.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>4.5-12</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
**DEFINE**

**Description**
The DEFINE command defines mnemonic for the mnemonic table in memory. FRAMESM LAPD includes a syntax option for the DEFINE command in the form of ,I. This option allows you to specify that an I-field is permitted with a pre-defined or user-defined frame type. You cannot define an I-field for the link access frames: SABM, SABME, DISC, DM, and UA.

**Type**
Statement or Direct

**Syntax**
DEFINE "name",I = x

*where: name* is a mnemonic name, one to six characters in length, and x is a numeric expression, of which the lower eight bits are significant. You can enter x in decimal or hexadecimal. To define the value in hexadecimal, precede the value with the ampersand (&) symbol.

The I option, if included, permits an I-field for that specific mnemonic. If the I is not entered as part of the command, an I-field is not permitted.

**Abbreviation**
DEFI.

**See Also**
DELETE, LMLIST, MLOAD, MSAVE

**Example**
This example defines a mnemonic called NEW with I-field permission.

DEFINE "NEW",I = 143
MLIST

(result) The mnemonic table is displayed, with the entry:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>I-Field</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>I</td>
<td>143</td>
<td>8F</td>
<td>10001111</td>
</tr>
</tbody>
</table>

**Example 2**
To remove an I-field option from the mnemonic, re-define the mnemonic without the I option, as shown below.

DEFINE "NEW" = 143
MLIST

(result) The NEW mnemonic is redefined, and appears in the mnemonic table as:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>I-Field</th>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td></td>
<td>143</td>
<td>8F</td>
<td>10001111</td>
</tr>
</tbody>
</table>
### FILL

**Description**
The FILL command changes the interframe fill pattern. This is the fill pattern, either 7E or FF, that is transmitted when no other data is being transmitted over the line.

The interframe fill can also be set using the FRAMEM LAPD parameter set-up menu. The menu selection for interframe fill is overridden by the FILL command.

**Type**
Statement or Direct

**Syntax**

\[
\text{FILL} = x
\]

where: \( x \) is either 7E or FF hexadecimal. A value of 7E indicates that the line is busy and the equipment is idle. A value of FF indicates that the line is idle.

**Abbreviation**
None

**See Also**
MENU

**Example**
This example causes the interframe fill to change repeatedly between FF and 7E, causing the Chameleon Data LED’s to flicker.

```
10 FILL = FF
20 FILL = 7E
30 GOTO 10
!RUN <RETURN>
```
RXCR

Description
RXCR is a variable that returns the value of the command/response bit of the received frame. It indicates whether the received frame is a command or a response. It has the following valid values:

<table>
<thead>
<tr>
<th></th>
<th>FROM NETWORK</th>
<th>FROM SUBSCRIBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RESPONSE</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

RXCR Values

Type
Variable

Syntax
(See example below.)

Abbreviation
RXC.

See Also
REC, RXFRLEN, TXCR

Example
10 REC
20 IF RXFRLEN = 0 GOTO 10
30 PRINT RXCR
!RUN <RETURN>
**RXSAPI**

**Description**
RXSAPI is a variable that returns the Service Access Point Identifier (SAPI) of the received frame.

**Type**
Variable

**Syntax**
See example below.

**Abbreviation**
RXS.

**See Also**
REC, RXFRLEN

**Example**
```
10 REC
20 IF RXFRLEN = 0 GOTO 10
30 PRINT RXSAPI
!RUN <RETURN>
```
RXTEI

Description  RXTEI is a variable that returns the Terminal End Point Identifier (TEI) of the received frame. This is used for extended addressing.

Type  Variable

Syntax  (See example below.)

Abbreviation  RXT.

See Also  REC, RXFRLEN, SET

Example  
10 REC  
20 IF RXFRLEN = 0 GOTO 10  
30 PRINT RXTEI  
!RUN  <RETURN>
TPRINT

Description
The TPRINT command displays the contents of the trace buffer, which contains the received and transmitted traffic.

The format of the trace buffer display is protocol-specific as shown in the figure below. For more information about the use of the trace buffer in simulation, refer to Section 3.2.

Type
Statement or Direct

Syntax
TPRINT

Abbreviation
TP.

See Also
TSAVE, TLOAD, LTPRINT, REC

Example 1
This is an example of a FRAMEM LAPD trace buffer display.
TXCR

Description
TXCR sets the value of the command/response bit of the frame being transmitted.

The value of this variable indicates whether the frame is a command or a response, from a network or a subscriber. It has the following valid values:

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>SUBSCRIBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td>1</td>
</tr>
<tr>
<td>RESPONSE</td>
<td>0</td>
</tr>
</tbody>
</table>

TXCR Values

Type
Variable

Syntax
(See example below.)

Abbreviation
TXC.

See Also
TRAN, TXFCTL

Example
10 TXCR = 1
20 TRAN
!RUN <RETURN>
**TXSAPI**

**Description**
TXSAPI sets the value of the Service Access Point Identifier (SAPI) for the frame being transmitted.

**Type**
Variable

**Syntax**
(See example below.)

**Abbreviation**
TXS.

**See Also**
TRAN, TXFCTL, SET

**Example**
10 TXSAPI = RXSAPI
20 TRAN
!RUN <RETURN>
TXTEI

Description  TXTEI is a variable that defines the Terminal End Point Identifier (TEI) for the frame being transmitted. It is used with extended addressing.

Type  Variable

Syntax  (See example below.)

Abbreviation  TXT.

See Also  TRAN, TXFCTL

Example  

10 TXTEI = 03  
20 TRAN  
!RUN  <RETURN>
FRAMEM LAPD SAMPLE PROGRAMS

FLAPD1

The following sample program is specific to FRAMEM LAPD. It builds and displays LAPD frames.

10  FLUSH          Clears the program buffer.
20  CLEAR          Clears trace buffer.
30  EXTEND         Sets extended addressing.
40  MOD128         Sets modulo to 128.
60  TXSAPI=16      Assigns a SAPI for the frame.
70  TXTEI=03       Assigns a TEI for the frame.
80  TXCR=1         Sets the C/R bit to 1.
90  TXP/F=1        Sets the poll/final bit to 1.
100 TRAN           Transmits the frame.
110 TXFCTL=IFRAME  Start building an I-frame.
120 TXM(R)=1       Sets N(R) to 1.
130 TXM(S)=3       Sets N(S) to 3.
140 TXP/F=0        Sets the poll/final bit to 0
150 TXIFIELD=ASC>IFI Assigns an ASCII value to the I-field of the frame.
160 BADTRAN        Transmits frame with a bad CRC.
170 TPRINT         Display contents of trace buffer.

RUN <RETURN>

The result of the TPRINT command is shown below. The first line is the frame transmitted by the TRAN command in line 100. The second line is the frame transmitted by the BADTRAN command in line 160 (shows a ? for the bad CRC).

T 00:28:04:60:10  4 C/R=01 SAPI=10 TEI=03 P IFRAME N(S)=0 N(R)=0
T ? 00:28:04:60:10  7 C/R=01 SAPI=10 TEI=03 IFRAME N(S)=3 N(R)=1 49 46 49
FLAPD2

The following sample program receives frames and determines whether or not they are valid commands or response frames.

10 DEFINE "NET" = 1

20 REC

30 IF RXFRLEN = 0 GOTO 20

40 IF CRC = 1 GOTO 20

50 IF RXCR = NET GOTO 140

60 IF RXFCTL = RR GOTO 240

70 IF RXFCTL = RNR GOTO 240

80 IF RXFCTL = REJ GOTO 240

90 IF RXFCTL = FRMR GOTO 240

100 IF RXFCTL = UA GOTO 240

110 IF RXFCTL = OM GOTO 240

120 PRINT "INVALID RESPONSE"

130 GOTO 20

140 IF RXFCTL = SABM GOTO 220

150 IF RXFCTL = SABNE GOTO 220

160 IF RXFCTL = DISC GOTO 220

170 IF RXFCTL = UI GOTO 220

180 IF RXFCTL = XID GOTO 220

190 IF RXFCTL = IFRAME GOTO 220

200 PRINT "INVALID COMMAND"

210 GOTO 20

220 REM Display message to command

230 REM...

240 REM Display message to response

250 REM...
4.6 FRAMEM DMI

Introduction

This section contains specific information about FRAMEM DMI. FRAMEM DMI includes special commands and variables that allow you to test the Digital Multiplexed Interface (DMI) using the Chameleon in conjunction with TEKELEC's TE820A T1/DS1 Frame Simulator/Analyzer.

Using FRAMEM DMI, you can configure your Chameleon to look like a host or a PBX, and simulate DMI Mode 0 communications.

This section assumes that you are familiar with the DMI interface and its operation. For additional information, refer to the appropriate AT&T publication.

Remember that in addition to the commands and variables described in this section, you can also use the BASIC commands described in Section 3.5 and the FRAMEM commands described in Section 4.3.

Setting Up

When you select FRAMEM DMI from the Chameleon main menu, the following screen is displayed:

Figure 4.6-1: DMI Screen
1. To run FRAMEM DMI immediately, press the ESC key.

2. To access the on-line help for setup procedures, press any key (other than ESC) when the DMI FRAMEM screen is displayed. This displays on-line documentation for setting up the DMI-BOS dialog. For additional information, refer to the Tekelec 820A User's Manual.

There are three consecutive screens that describe the equipment you will need for your simulation or analysis application. These screens also provide detailed test setup instructions.

The bottom of the screen gives you instructions for moving from screen to screen within DMI setup. You can use the following keys:

- Press F to move forward to the next screen
- Press B to move back to the previous screen
- Press ESC to run FRAMEM DMI

3. When you are ready to run FRAMEM DMI, press ESC and the following message is displayed:

DMI FRAMEM V X.X Copyright 1987 TEKELEC

This gives you the opportunity to check the software version you are running, where x.x.x is the version number.

4. The FRAMEM simulation prompt (!) appears, indicating that the system is ready to accept commands.

The FRAMEM DMI commands are listed on the following pages. The first page lists the commands by function. Following that, the commands are listed in alphabetical order, one command per page.
The commands that are available only in FRAMEM DMI are described in this section. This page lists the FRAMEM DMI commands and variables in tables by function. For a complete description of a command, refer to the page number indicated.

If you are using FRAMEM, you should also read Section 4.3, which contains the commands that are available for all FRAMEM subsets, and Section 3.5, which lists the Chameleon BASIC commands.

### TABLE 4.6-2: FRAMEM DMI READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAUSE</td>
<td>4.6-5</td>
<td>Variable</td>
<td>Returns cause of an on-hook state.</td>
</tr>
<tr>
<td>DCALLED</td>
<td>4.6-8</td>
<td>Variable</td>
<td>Returns phone number to be called.</td>
</tr>
<tr>
<td>DCALLING</td>
<td>4.6-9</td>
<td>Variable</td>
<td>Returns phone number of received call.</td>
</tr>
<tr>
<td>DMATCH</td>
<td>4.6-11</td>
<td>Variable</td>
<td>Returns phone number that will be accepted for incoming calls in Wink mode.</td>
</tr>
<tr>
<td>GLARE</td>
<td>4.6-15</td>
<td>Variable</td>
<td>Indicates if a glare condition exists.</td>
</tr>
<tr>
<td>RESPTIME</td>
<td>4.6-19</td>
<td>Variable</td>
<td>Determines how busy the switch is.</td>
</tr>
<tr>
<td>STATE</td>
<td>4.6-20</td>
<td>Variable</td>
<td>Returns the state of a call and the operating mode.</td>
</tr>
<tr>
<td>STATUS</td>
<td>4.6-21</td>
<td>Variable</td>
<td>Returns state of the call, call setup, modulo, addressing, and whether a glare condition exists.</td>
</tr>
</tbody>
</table>
## TABLE 4.6-3: FRAMEM DMI COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHADMIN</td>
<td>4.6-6</td>
<td>S or D</td>
<td>Defines the call setup mode to use.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>4.6-7</td>
<td>S or D</td>
<td>Performs call setup procedures and seizes selected channel on the TE820A.</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>4.6-10</td>
<td>S or D</td>
<td>Causes Chameleon/TE820A to go on-hook.</td>
</tr>
<tr>
<td>DTIMERS</td>
<td>4.6-12</td>
<td>S or D</td>
<td>Sets the value of one of the eight DMI timers.</td>
</tr>
<tr>
<td>MATCH</td>
<td>4.6-16</td>
<td>S or D</td>
<td>Defines the phone number that will be accepted for incoming calls Wink mode.</td>
</tr>
<tr>
<td>OUTNUM</td>
<td>4.6-17</td>
<td>S or D</td>
<td>Defines the number to be dialed.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>4.6-22</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
CAUSE

Description
CAUSE is a read-only variable that determines the cause of an on-hook state. The valid values and their associated meanings are:

0  Start of simulation
1  Call rejected, no match on address digits
2  No wink received before T1 timeout
3  No off-hook (incoming) before T7 timeout
4  Local disconnect
5  Remote disconnect

Type
Read-only Variable

Syntax
(See example below.)

Abbreviation
C.

See Also
STATUS

Example
PRINT CAUSE

(result)  4  (The on-hook state was caused by a local disconnect.)
CHADMIN

Description
CHADMIN defines which call setup mode to use. You can only change CHADMIN when you are in a disconnected state. Use the commands PRINT CHADMIN or STATUS to determine the current value of CHADMIN.

Type
Statement or Direct

Syntax
CHADMIN = x

where: x is a number between 0 and 3 that represents the call setup mode options, as follows:

0 = Wink-start in/Wink-start out
1 = Auto-start in/Wink-start out
2 = Wink-start in/Auto-start out
3 = Auto-start in/Auto-start out

These setup modes are determined using Table 4.6-3.

<table>
<thead>
<tr>
<th>BIT #</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1(IN)</th>
<th>0(IN)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

X = DON'T CARE
0 = WINK-START
1 = AUTO-START

Figure 4.6-4: Setup Mode Bit Values

Abbreviation
CHAD.

See Also
STATUS

Example
10 CHADMIN=2
20 STATUS
!RUN

(result) DISCONNECTED START OF SIMULATION
WINK-START-IN AUTO-START-OUT
Modulo8/Normal addressing
CONNECT

Description
The CONNECT command performs call setup procedures and seizes the channel selected on the TE820A. If required, it also transmits the dial pulses that are defined by the OUTNUM variable. It refers to bit zero of the CHADMIN variable to determine the appropriate setup mode.

Type
Statement or Direct

Syntax
CONNECT

Abbreviation
CON.

See Also
OUTNUM, CHADMIN

Example
CONNECT
DCALLED

Description
The DCALLED command displays the phone number to be outpulsed (dialed) when a CONNECT command is executed. The outpulsed number is assigned using the OUTNUM command.

Type
Read-Only Variable

Syntax
DCALLED

Abbreviation
DC.

See Also
CONNECT, OUTNUM, DCALLING

Example
DCALLED
(result) NUMBER TO BE CALLED = 8188805656
DCALLING

Description  The DCALLING command displays the numbers inputted (received).

Type  Read-only Variable

Syntax  DCALLING

Abbreviation  DCALLI.

See Also  DCALLED

Example  DCALLING

(result)  NUMBER RECEIVED =8188805656
## DISCONNECT

<table>
<thead>
<tr>
<th>Description</th>
<th>DISCONNECT causes the Chameleon/TE820A to go on-hook. The DISCONNECT command resets the acquisition buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>DISCONNECT</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>DI.</td>
</tr>
<tr>
<td>See Also</td>
<td>CONNECT</td>
</tr>
<tr>
<td>Example</td>
<td>DISCONNECT</td>
</tr>
</tbody>
</table>
DMATCH

Description  DMATCH displays the number the Chameleon will accept for incoming calls in Wink mode. The number is defined using the MATCH variable.

Type  Read-only Variable

Syntax  DMATCH

Abbreviation  DM.

See Also  DCALLING

Example 1  

```
10 MATCH "8188805656"
20 DMATCH
!RUN  <RETURN>
```

(result)  NUMBER TO BE MATCHED = ANY NUMBER IS ACCEPTABLE

Example 2  

This example defines the number to accept for incoming calls and then displays the number using DMATCH.

```
10 MATCH "8188805656"
20 DMATCH
!RUN  <RETURN>
```

(result)  NUMBER TO BE MATCHED = 8188805656


## DTIMERS

### Description
The DTIMERS command displays the current timer settings. FRAMEM DMI has eight timers that you can program while you are in a disconnected state. These timers are:

<table>
<thead>
<tr>
<th>TIMER</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Time between an incoming seizure and the start of the outgoing wink, or the maximum time allowed in waiting for an incoming wink.</td>
</tr>
<tr>
<td>T2</td>
<td>Duration of the wink signal.</td>
</tr>
<tr>
<td>T3</td>
<td>Time between the end of the wink signal and the first dial pulse.</td>
</tr>
<tr>
<td>T4</td>
<td>The duration of a break pulse (used in dialing).</td>
</tr>
<tr>
<td>T5</td>
<td>The duration of a make pulse (used in dialing).</td>
</tr>
<tr>
<td>T6</td>
<td>Inter-digit time (used in dialing).</td>
</tr>
<tr>
<td>T7</td>
<td>Dialing timeout. The time between the last dial pulse and the issue of an outgoing offhook, or the maximum time allowed in waiting for an incoming offhook signal once the last dial pulse is sent.</td>
</tr>
<tr>
<td>T8</td>
<td>Minimum disconnect time.</td>
</tr>
</tbody>
</table>

Figure 4.6-5: FRAMEM DMI Timers

Figures 4.6-6, 4.6-7, and 4.6-8 illustrate the time periods associated with the eight DMI timers.

### Type
Read-only Variable

### Syntax
DTIMERS

To set a timer value, use the following syntax:

\[
T_x = y \]

where: \( x \) is the timer number in the range 1 - 8 and \( y \) is the timer value in the range 1 - 99. T1 - T6 and T8 are in units of tens of milliseconds (.0001). T7 is in units of seconds.

### Abbreviation
DT.
Example

This example sets values for the T1 and T2 timers and then displays the values of all timers.

```
10 T1=15
20 T2=25
30 DTIMERS
!RUN <RETURN>
```

(result)

<table>
<thead>
<tr>
<th>Timer</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Time between seizure &amp; start of wink</td>
<td>15</td>
</tr>
<tr>
<td>T2</td>
<td>Duration of wink signal</td>
<td>25</td>
</tr>
<tr>
<td>T3</td>
<td>Time between end of wink &amp; first dial pulse</td>
<td>07</td>
</tr>
<tr>
<td>T4</td>
<td>Duration of break pulse</td>
<td>06</td>
</tr>
<tr>
<td>T5</td>
<td>Duration of make pulse</td>
<td>04</td>
</tr>
<tr>
<td>T6</td>
<td>Inter-digital interval</td>
<td>60</td>
</tr>
<tr>
<td>T7</td>
<td>Dialing time-out interval (seconds)</td>
<td>10</td>
</tr>
<tr>
<td>T8</td>
<td>Minimum disconnect time</td>
<td>42</td>
</tr>
</tbody>
</table>

Figure 4.6-6: Wink (Normal)
Figure 4.6-7: Wink (Abnormal) Time Period

Figure 4.6-8: Auto (Normal) Time Period

*In auto-start, if no off-hook is received after T1, disconnect*
GLARE

Description GLARE is a read-only variable that determines whether a glare condition exists. This variable is relevant only when the system is in a CONNECTED state. The Chameleon automatically handles glare resolution according to the DMI specifications.

GLARE has the following values:
GLARE = 0  No glare condition exists.
GLARE = 1  Glare condition exists.

The GLARE variable is reset to 0 after DISCONNECT.

Type       Read-only Variable
Syntax     (See example below.)
Abbreviation G.
See Also   STATUS
Example 1  PRINT GLARE
(result)   0

Example 2  IF GLARE = 0 PRINT "NO GLARE CONDITION"
MATCH

Description MATCH specifies which incoming calls will be accepted. It is only relevant when you have wink start-in mode.

Type Statement or Direct

Syntax MATCH = "x"

where:  x is a valid incoming telephone number.

To accept any call enter double quotation marks without specifying a number:

MATCH = ""

Abbreviation MA.

See Also DMATCH

Example 1 This example assigns 880-5656 as the number which will be accepted for incoming calls.

10 MATCH="8805656"
20 DMATCH
!RUN  <RETURN>

(result) NUMBER TO BE MATCHED = 8805656

Example 2 This example allows any number to be accepted for incoming calls.

10 MATCH = ""
20 DMATCH
!RUN  <RETURN>

(result) NUMBER TO BE MATCHED = ANY NUMBER IS ACCEPTABLE
OUTNUM

Description
OUTNUM sets the number to be outpulsed (dialed).

Type
Statement or Direct

Syntax
OUTNUM = "x"

$A = "x"
OUTNUM = $A

where: x is the phone number to be dialed, with a maximum of 30 digits in length.

Abbreviation
OU.

See Also
DCALLED

Example
10 OUTNUM="8805656"
20 DCALLED
!RUN <RETURN>

(result) NUMBER TO BE CALLED =8805656
# RESET

<table>
<thead>
<tr>
<th>Description</th>
<th>The RESET command clears the acquisition buffer, the buffer that receives data from the line. It also resets the state of the call to its start of simulation disconnected state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>RESET</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>None</td>
</tr>
<tr>
<td>See Also</td>
<td>CLEAR, REC</td>
</tr>
<tr>
<td>Example</td>
<td>RESET</td>
</tr>
</tbody>
</table>
RESPTIME

Description
RESPTIME is a read-only variable that determines how busy your switch is. In wink-start mode, it displays the time between an outgoing seizure and the start of an incoming wink signal. In Auto-Start mode, it displays the time from the outgoing seizure and the incoming off-hook.

Type
Read-only Variable

Syntax
(See example below.)

Abbreviation
RE.

Example 1
PRINT RESPTIME

Example 2
IF RESPTIME>50 PRINT "SLOW RESPONSE TIME."
STATE

Description
STATE is a read-only variable that determines the state of a call (connected or disconnected) and the operating mode (wink-start in/wink-start out, auto-start in/wink-start out, wink-start in/auto-start out or auto-start in/auto-start out).

The valid values for STATE and their meaning are:

1 Disconnected (on-hook) xxxxxx (xxxxxx is the cause)
2 Outgoing setup (i.e. exchanging winks)
3 Incoming setup
4 Dial pulses being received
5 Dial pulses being sent
6 Connected

You can display the state of the call and the call setup mode using the STATUS.

Type
Read-only Variable

Syntax
(See examples below.)

Abbreviation
ST.

See Also
STATUS

Example 1
PRINT STATE
(result) 5

Example 1
IF STATE = 6 PRINT "CALL CONNECTED"
(result) CALL CONNECTED
**STATUS**

**Description**
STATUS displays the state of the call, the call setup mode, the modulo (8 or 128), and type of addressing (normal or extended). It also indicates if a glare condition exists.

**Type**
Read-only Variable

**Syntax**
STATUS

**Abbreviation**
STA.

**See Also**
STATE

**Example**

```
!STATUS
```

(result)

```
DISCONNECTED NO OFF HOOK BEFORE T7
WINK-START-IN AUTO -START-IN
Modulo8/Normal Addressing
```
TPRINT

Description
TPRINT displays the contents of the trace buffer. The format of the trace buffer display is shown below.

Type
Statement or Direct

Syntax
TPRINT

Abbreviation
TP.

See Also
TSAVE, TLOAD

Example
!TPRINT
(result)
FRAMEM DMI SAMPLE PROGRAMS

Program DMI1

This program sets up and transmits a call, using the OUTNUM command.

10 $A = "123456"  
Assigns number to $A.

20 OUTNUM = $A  
Specifies $A as number to be outpulsed.

30 CONNECT  
Seizes channel and transmits the dial pulses defined above.

40 IF STATE#6 GOTO 40  
Stays here until connected.

50 TXFCTL = SABM  
Defines SABM to be transmitted.

60 TRAN  
Transmits the frame.
Program DMI2

This program places a call and sends a string of decimal numbers to be transmitted down the line.

10 FOR A = 0 TO 9
20 $A = **
30 FOR B = 1 TO 9
40 $A = $A+DECS(A)
50 NEXT B
60 OUTNUM = $A
70 GOSUB 1000
80 NEXT A
90 STOP
1000 CONNECT
1010 IF STATE#6 GOTO 1010
1020 DISCONNECT
1030 RETURN

Sets up loop A to repeat 10 times, to generate 10 telephone numbers.
Initializes string variable $A.
Sets up loop B to repeat A 9 times to generate a 9-digit phone number.
Assigns number to $A.
Increments loop B counter.
Number to be outputted is $A.
Executes subroutine at line 1000.
Increments A to generate a different telephone number.
Stops program when A > 9.
Seizes a channel and transmits.
Waits until connected.
Disconnects call.
Ends subroutine. Returns to line 80.

Program: DMI3

This program checks and displays the status of the line.

10 A=0
20 CLS
30 IF A = STATE GOTO 30
40 A = STATE
50 XYPL0T(10,1)
60 ERAE0L
70 XYPL0T(9,1)
80 STATUS
90 GOTO 30

Sets A to 0.
Clears the screen.
If state doesn't change, stays at 30.
Sets A to the value of state.
Positions cursor on line 10.
Erases to end of line.
Positions cursor on line 9.
Displays state of line.
Determines if the state has changed.
Program DMI4

This program accepts and displays incoming numbers.

```
10 A = 0
20 CLS
30 IF STATE=4 GOTO 70
40 IF A=STATE GOTO 40
50 A=STATE
60 GOSUB 100
70 GOSUB STATE*1000
80 GOTO 30
100 XYPLOT(5,1)
110 ERAEOL
120 RETURN
1000 XYPLOT(5,5)
1010 PRINT "Disconnected"
1020 GOTO 3020
2000 XYPLOT(5,5)
2010 PRINT "OUTGOING SETUP"
2020 RETURN
3000 XYPLOT(5,5)
3010 PRINT "INCOMING SETUP"
3020 XYPLOT(7,0)
3030 ERAEOL
3040 RETURN
4000 XYPLOT(5,5)
4010 PRINT "RECEIVING DIAL PULSES"
4020 XYPLOT(6,1)
4030 DCALLING
4040 RETURN
5000 XYPLOT(5,5)
5010 PRINT "SENDING DIAL PULSES"
5020 XYPLOT(6,1)
5030 DCALLED
5040 RETURN
6000 XYPLOT(5,5)
6010 PRINT "CONNECTED"
6020 RETURN
```

Sets A to 0.
Clears the screen.
If dial pulse being received, go to 70.
If state doesn't change, stays at 40.
Sets A to the value of state.
Executes subroutine starting at 100.
Execute subroutine depending on state of call.
After subroutine, returns to this line which goes to line 30.
Positions cursor on line 5.
Erases text on line 5.
Ends subroutine.
Positions cursor on line 5, column 5.
Prints message
Goes to line 3020.
Positions cursor on line 5, column 5.
Prints message
Ends subroutine.
Positions cursor on line 5, column 5.
Prints message.
Positions cursor on line 7, column 0.
Erases text on line 7.
Ends subroutine.
Positions cursor on line 5, column 5.
Prints message.
Positions cursor on line 6, column 1.
Displays the inputed number.
Ends subroutine.
Positions cursor on line 5, column 5.
Prints message.
Positions cursor on line 6, column 1.
Displays the outputed number.
Ends subroutine.
Positions cursor on line 5, column 5.
Prints message.
Ends subroutine.
5.1 INTRODUCTION TO SIMP/L

Introduction

Simulated Interactive Multi-Protocol Language (SIMP/L) is a highly flexible programming language designed for engineers working in SNA/SDLC, X.25/HDLC, LAPD, or developing non-standard hybrid protocols.

SIMP/L Function

SIMP/L allows you to create mnemonic tables with automatic level 1 and 2 support. The sophisticated SIMP/L software handles frame level specifications automatically, including:

- Address and control fields
- CRC generation
- Lower level physical interface
- Link control
- Frame generation and reception
- Command and response generation

Link Level

You can use SIMP/L to emulate any protocol which uses an HDLC, SDLC, or LAPD link level. You configure the automatic link level by using the parameter set-up menu and SIMP/L commands. This allows you to concentrate on higher levels without investing large amounts of time on the link level.

Using SIMP/L

Sections 5.2 and 5.3 describe the commands and variables that are available for all SIMP/L protocol subsets. For this reason, you should read Sections 5.2 and 5.3 regardless of the protocol you are using.

In addition there are three protocol subsets of SIMP/L that make use of commands and variables that are specific to the protocol. You should also read the section that corresponds to the protocol you are using, as follows:

- Section 5.4 - SIMP/L HDLC
- Section 5.5 - SIMP/L SDLC
- Section 5.6 - SIMP/L LAPD
- Section 5.7 - Multi-Link SIMP/L LAPD
- Section 5.8 - SIMP/L V.120

Remember that the Chameleon BASIC commands and variables are part of SIMP/L. These are described in Section 3.5.
Flow Control  

Flow control occurs at the frame level, and depends on the amount of buffer space available on the Chameleon to receive data. If the receive buffer becomes full, the Chameleon declares the station busy and sends an RNR to the device under test. When the buffer is cleared, the Chameleon sends an RR to indicate that the busy condition has cleared.

To keep the receive buffer clear and avoid a station busy condition from disrupting your test, it is recommended that your program include a number of REC commands.
5.2 SIMP/L COMMAND INDEX

Introduction

This section lists by function the commands and variables that are common to all protocol subsets of SIMPL. For more information about a particular SIMP/L command, refer to the page number indicated.

There are also commands and variables that are specific to each protocol subset of SIMP/L. Refer to the appropriate section for a description of these.

### TABLE 5.2-1: SIMP/L MISCELLANEOUS COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE</td>
<td>5.3-5</td>
<td>S or D</td>
<td>Defines a mnemonic for the mnemonic table in memory.</td>
</tr>
<tr>
<td>SET</td>
<td>5.3-12</td>
<td>S or D</td>
<td>Assigns values to protocol-specific timers and parameters.</td>
</tr>
<tr>
<td>SLOF</td>
<td>5.3-13</td>
<td>S or D</td>
<td>Sets the link off.</td>
</tr>
<tr>
<td>SLON</td>
<td>5.3-14</td>
<td>S or D</td>
<td>Sets the link on.</td>
</tr>
</tbody>
</table>

### TABLE 5.2-2: SIMP/L PRINT AND DISPLAY COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRDISPf</td>
<td>5.3-8</td>
<td>S or D</td>
<td>Outputs the last data field received to a printer or remote device.</td>
</tr>
<tr>
<td>LTDISPf</td>
<td>5.3-9</td>
<td>S or D</td>
<td>Outputs the last data field built to a printer or remote device.</td>
</tr>
<tr>
<td>RDISPF</td>
<td>5.3-10</td>
<td>S or D</td>
<td>Displays the last data field received.</td>
</tr>
<tr>
<td>TDISPf</td>
<td>5.3-16</td>
<td>S or D</td>
<td>Displays the last data field built.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.3-17</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
### TABLE 5.2-3: SIMP/L TRANSMISSION AND RECEPTION COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAK</td>
<td>5.3-2</td>
<td>S or D</td>
<td>Breaks a message into mnemonics and/or strings.</td>
</tr>
<tr>
<td>BUFFER</td>
<td>5.3-3</td>
<td>S or D</td>
<td>Assembles a message in hex.</td>
</tr>
<tr>
<td>BUILD</td>
<td>5.3-4</td>
<td>S or D</td>
<td>Assembles a message from mnemonics and/or strings.</td>
</tr>
<tr>
<td>REC</td>
<td>5.3-11</td>
<td>S or D</td>
<td>Receives a message.</td>
</tr>
<tr>
<td>TRAN</td>
<td>5.3-18</td>
<td>S or D</td>
<td>Transmits a message.</td>
</tr>
</tbody>
</table>

### TABLE 5.2-4: SIMP/L READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH</td>
<td>5.3-6</td>
<td>Variable</td>
<td>Returns the length of the received frame.</td>
</tr>
<tr>
<td>LNKSTAT</td>
<td>5.3-7</td>
<td>Variable</td>
<td>Returns a value that indicates the status of the link.</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.3-15</td>
<td>Variable</td>
<td>Returns status of the link, timer values, and protocol-specific parameter settings.</td>
</tr>
</tbody>
</table>
5.3 SIMP/L COMMANDS

Introduction

This section provides a complete description of the SIMP/L commands. The commands are organized alphabetically, one command per page.

If you know the name of the command you want to use, you can look up the command in this section. If you are unsure of the command you need, refer to the previous section which lists the commands by function.

The following information is provided in this section:

• Command description
• Type of command (Statement or Direct)
• Syntax
• Command abbreviation
• Related commands (See Also)
• Example(s) of command usage
### BREAK

**Description**
BREAK disassembles an I-field into its component strings and/or user-defined mnemonics.

**Type**
Statement or Direct

**Syntax**

- `BREAK udm,udm,udm`
- `BREAK $A,$B,$C...`
- `BREAK udm,$A,$B,udm,$C...`

*where*: udm is a user-defined mnemonic. Refer to Chapter 3.2 for general information about the use of mnemonics. Refer to the protocol chapter for a description of the mnemonic table for a particular SIMP/L protocol.

**Abbreviation**
BR.

**See Also**
BUILD

**Example**
This example breaks an I-field into General Format Identifier (GFI), Logical Channel Group Number (LCGN), Logical Channel Number (LCN) and Packet Identifier (PKIDNT).

```plaintext
10 GFI=4
20 LCGN=4
30 LCN=8
40 PKID=8

50 SLOF

60 REC
70 IF LENGTH=0 GOTO 60
80 BREAK GFI,LCGN,LCN,PKID
90 PRINT GFI,LCGN,LCN,PKID
100 SLOF
```

Lines 10-40 defines the mnemonics that will be used to disassemble the frame.

Sets the link on to establish link level.

Attempts to receive a frame.

Loops until an I-frame is received.

Disassembles the received frame using the defined mnemonics.

Displays the mnemonic values.

Disconnects the link.
BUFFER

Description
BUFFER defines a message for the transmission buffer in hexadecimal code.

Type
Statement or Direct

Syntax
BUFFER = xxx

where: xxx is the message. You can enter blank spaces between every two characters without affecting the transmission in hexadecimal. Spaces added elsewhere cause single digits to be interpreted independently (see Example 2 below).

Abbreviation
BUF.

See Also
TRAN, TPRINT

Example 1
To assign 12 34 56 78 09 to the transmission buffer, you can enter either of the examples below:

BUFFER = 123456789
BUFFER = 12 34 56 78 9

Example 2
To transmit 01 02 34 56 78 9, you would enter:

10 BUFFER = 1 2 34 56 78 9 Defines the message to be transmitted.
20 TRAN Passes the contents of the buffer to link level for transmission.
## BUILD

**Description**
BUILD assembles a message in the transmission buffer using the information defined in strings and/or user-defined mnemonics.

Refer to Section 3.2 for general information about using mnemonics. Refer to the appropriate section in this chapter for a description of the mnemonic table for a particular SIMP/L protocol.

**Type**
Statement or Direct

**Syntax**
`BUILD udm, udm, udm ...
BUILD $A, $B, $C ...
BUILD udm, $A, $B, udm, $C ...

where: udm is a user-defined mnemonic.`

**Abbreviation**
BU.

**See Also**
BREAK

**Example**
This example builds a message using two strings.

```
10 $A = "HELLO"  Assigns a value to string $A.
20 $B = "GOODBYE" Assigns a value to string $B.
30 BUILD $A, $B  Builds a message using $A and $B.
40 TRAN  This will pass the string HELLOGOODBYE to the link level for transmission.
```
### DEFINE

**Description**

DEFINE defines a new frame control mnemonic or redefines an existing mnemonic. Refer to Section 3.2 for a general description of the use of mnemonics in simulation. Refer to the appropriate section in this chapter for a description of the default mnemonic table for the protocol you are using.

**Type**

Statement or Direct

**Syntax**

```plaintext
DEFINE"name" = x
```

where: `name` is a mnemonic name of one to six characters and `x` is any valid expression that defines the field width in bits. The maximum value of `x` is 16.

**Abbreviation**

DEF.

**See Also**

DELETE, MLIST, MSAVE, MLOAD

**Example**

This example defines a mnemonic named NEW with a length of 16 bits. The table is then displayed to verify the new entry.

```plaintext
10 DEFINE"NEW" = 16
20 MLIST
```

The mnemonic table is displayed, including the entry:

```
NEW     16
```

The mnemonic can then be used to assemble messages for transmission or disassemble received messages. Refer to the Example for the BREAK command on page 5.3-2.
LENGTH

Description
LENGTH is a read-only variable that returns the length of the received frame. It is updated following the REC command. If no frame is available, the length is zero.

Type
Variable

Syntax
(See examples below.)

Abbreviation
LENG.

See Also
REC

Example 1
This example displays the current value of LENGTH.
PRINT LENGTH

Example 2
This example attempts to receive a frame. If no frame is available, it loops to receive again.

10 REC
20 IF LENGTH = 0 GOTO 10
30 REM The following statements process the received data
40...
50...
LNKSTAT

Description

LNKSTAT is a read-only variable that returns a value that indicates the current status of the link. The status indicated by the value is protocol-specific. For more information, refer to the SIMP/L section that corresponds to the protocol you are using.
## LRDISPF

**Description**
LRDISPF outputs the last data field received to a printer or remote device. The last data field received is stored in a special reception buffer. For more information about simulation buffers, refer to Section 3.2.

**Type**
Statement or Direct

**Syntax**
LRDISPF

**Abbreviation**
LRD.

**See Also**
RDISPF, TDISPF, LTDISPF

**Example**
LRDISPF
LTDISPF

Description  LTDISPF outputs the last data field built to a printer or remote device. The last data field built is stored in a special transmission buffer. For more information about simulation buffers, refer to Section 3.2.

Type  Statement or Direct

Syntax  LTDISPF

Abbreviation  LTD.

See Also  TDISPF, RDISPF, LRDISPF

Example  LTDISPF
# RDISPF

**Description**
RDISPF displays the last data field received. The last data field received is stored in a special reception buffer. For more information about simulation buffers, refer to Section 3.2.

**Type**
Statement or Direct

**Syntax**
RDISPF

**Abbreviation**
RDIS.

**See Also**
LRDISPF, TDISPF, LTDISPF

**Example**
RDISPF
REC

Description
REC transfers the next message in sequence from the reception buffer to the trace buffer. It also updates the read-only variable LENGTH, which returns the length of the received message.

Refer to Section 3.2 for a description of simulation buffers.

Type
Statement or Direct

Syntax
REC

Abbreviation
None

See Also
TPRINT, RDISPF, LRDISP, LENGTH

Example
This example receives a frame, disassembles it using mnemonics, and prints the mnemonic values.

```
10 GFI=4
20 LCGN=4
30 LCN=8
40 PKIDNT=8
50 SLON
60 REC
70 IF LENGTH=0 GOTO 60
80 BREAK GFI,LCGN,LCN,PKIDNT
90 PRINT GFI,LCGN,LCN,PKIDNT
100 SLOF
```

Lines 10-40 defines mnemonics for X.25 protocol elements.
Sets the link on to establish link level.
Attempts to receive a frame.
Loops until an I-frame is received.
Disassembles received frame using defined mnemonics.
Displays mnemonic values.
Disconnects the link.
SET

Description

The SET command sets the value of SIMP/L variables, counters, and timers. Since the variables, counters, and timers are protocol-specific, refer to the SIMP/L section that describes the protocol you are using for a description of the syntax.
SLOF

Description
The SLOF (Set Link Off) command disconnects the link by sending a disconnect (DISC) frame.

Type
Statement or Direct

Syntax
SLOF

Abbreviation
None

See Also
SLON

Example
This example sets the link on, receives a frame, disassembles the frame, and then sets the link off.

10 GFI=4
20 LCGN=4
30 LCN=8
40 PKIDNT=8
50 SLON
60 REC
70 IF LENGTH=0 GOTO 60
80 BREAK GFI,LCGN,LCN,PKIDNT
100 SLOF

 Lines 10-40 defines mnemonics for X.25 protocol elements.
 Sets the link on to establish link level.
 Attempts to receive a frame.
 Loops until an I-frame is received.
 Disassembles the received frame into the protocol elements.
 Disconnects the link.
## SLON

**Description**  
The SLON (Set Link On) command attempts to set the frame level link by sending a SABM (HDLC or LAPD), SABME (LAPD Mod 128), or SNRM (SDLC).

**Type**  
Statement or Direct

**Syntax**  
SLON

**Abbreviation**  
SL.

**See Also**  
SLOF

**Example**  
SLON
The STATUS command displays the current link status and the values of the variables, counters, and timers. Since the variables, counters, and timers are protocol-specific, refer to the SIMP/L section that describes the protocol you are using for a description of the STATUS display.
TDISPF

Description: The TDISPF command displays the last data field built. The last data field built is stored in a special transmission buffer. For more information about simulation buffers, refer to Section 3.2.

Type: Statement or Direct

Syntax: TDISPF

Abbreviation: TDIS.

See Also: LTDISPF, RDISPF, LRDISPF, BUILD

Example:

10 $A="HELLO"
20 $B="GODBYE"
30 BUILD $A,$B
40 TDISPF

Result: 48454C4C4F474F44425945
TPRINT

Description

The TPRINT command displays the contents of the trace buffer, which stores the transmitted and received traffic (following a REC command). The format of the display is protocol-specific. Refer to the appropriate SIMP/L for a description of the TPRINT display for the protocol you are using.

You can stop the list using CTRL S. Resume the list with CTRL Q. Abort using the ESC key.

The trace buffer display is protocol-specific. The general format of the trace buffer display is as follows:

```
T ? 00:00:00:00:00 xxxx ...
```

- **T** = Transmitted frame
- **R** = Received frame
- **?** = Bad CRC (frame overflow or abort)
- **Space** = Good CRC
- **T** = Time Stamp
- **00:00:00:00:00** = Hours:Minutes:Seconds:01Seconds:0001Seconds
- **xxxx** = First byte of frame
- **1 - 4 digits** indicating frame length (right-justified)

Type

Statement or Direct

Syntax

TPRINT

Abbreviation

TP.

See Also

TLOAD, TSAVE, LTPRINT

Example

(See protocol-specific sections for examples.)
TRAN

Description  The TRAN command transmits a message which was built using the BUILD command. The link must be established (using SLON) before you use TRAN.

Type  Statement or Direct

Syntax  TRAN

Abbreviation  TR.

See Also  BUILD, SLON

Example  
10 $A="HELLO"
20 BUILD $A
30 TRAN
5.4 SIMP/L HDLC

General Characteristics

The general characteristics for SIMP/L HDLC are as follows:

- Full duplex
- Bit rate up to 64 Kbps with clocks received or supplied
- Maximum frame length of 512 bytes
- Maximum number of outstanding frames is 7
- Line code is NRZ or NRZI
- Simulation of DCE or DTE
- CRC CCITT standards
- The frame level conforms to the CCITT X.25 standard of June 1980 (LAPB only)
- Emulate a network or subscriber

Flow Control

Flow control occurs at the frame level, and depends on the amount of buffer space available on the Chameleon to receive data. If the receive buffer becomes full, the Chameleon declares the station busy and sends an RNR to the device under test. When the buffer is cleared, the Chameleon sends an RR to indicate that the busy condition has cleared.

To keep the receive buffer clear and avoid a station busy condition from disrupting your test, it is recommended that your program include a number of REC commands.

There are eight possible states in the frame level which can be displayed using the STATUS command or the LNKSTAT command. The LNKSTAT command (page 5.4-6) describes each of these states.

HDLC Frame Format

Figure 5.4-1 illustrates the format of an HDLC frame with the field widths in bits indicated.

<table>
<thead>
<tr>
<th>FLAG</th>
<th>ADDRESS</th>
<th>CONTROL FIELD</th>
<th>I-FIELD</th>
<th>FRAME CHECK SEQUENCE</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8 or 16</td>
<td>8 or 16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.4-1: Frame Format
The opening and closing flags signal the end of the preceding frame and the beginning of the next frame, or is transmitted as fill time between multiple frames. A flag has the unique bit pattern: 0111 1110.

The **Address Field** contains the address of the station that is receiving a command or sending a response to a command.

The **Frame Check Sequence** (FCS) determines if the data in the packet was received without error. It is a 16-bit Cyclic Redundancy Check (CRC) that is calculated from the data.

The **Control Field** contains 8 bits which identify the type of frame being transmitted, as indicated in the figure below.

<table>
<thead>
<tr>
<th>Format</th>
<th>Commands</th>
<th>Responses</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information transfer</td>
<td>I (information)</td>
<td>0</td>
<td>N(S)</td>
</tr>
<tr>
<td>Supervisory</td>
<td>RR (receive ready)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RNR (receive not ready)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>REJ (reject)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Unnumbered</td>
<td>SARM (set asynchronous</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>response mode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SABM (set asynchronous</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>balanced mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DISC (disconnect)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>UA (unnumbered</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>acknowledgement)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMDR (command reject)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>FRMR (frame reject)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.4-2: HDLC Frame Formats
SIMP/L HDLC PARAMETER SET-UP MENU

Introduction

This section describes the parameters and valid values for the SIMP/L HDLC menu. This menu (see figure below) enables you to configure and save parameters for running SIMP/L HDLC simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu (option S), refer to Section 2.3.

The eight parameters and valid settings are listed below. Each parameter is described on the next page.

A. Simulate: DCE, DTE
B. Data Encoding: NRZ, NRZI
C. Bit Rate: Received (if DTE)
               Range: 50 - 64000 bits (if DCE)
D. Station Type: NETWORK, SUBSCRIBER
E. Frame Window: Range: 1 - 7 frames
F. T1: Range: 1 - 255 seconds
G. N1: Range: 1 - 255 retransmissions
H. N2: Range: 1 - 255 retransmissions

Figure 5.4-3: The SIMP/L HDLC Parameter Set-Up Menu
If you are unfamiliar with these parameters, use the following guidelines for assistance in selecting the correct settings.

**Simulate:**
**DCE/DTE**

The DCE furnishes the clock which determines the data rate. It also determines on which connector pins the data is physically sent and received. For Simulation, one device is designated as the DCE; the other device is designated as a DTE.

If the Unit Under Test (UUT) is supplying the clock, it is the DCE. In this case, the Chameleon must be configured as a DTE. If the UUT is not supplying the clock, it is designated as a DTE. The Chameleon must then be configured as the DCE.

**Data Encoding:**
**NRZ/NRZI**

This parameter determines if the data is transmitted in NRZ (Non-Return to Zero) or NRZI (Non-Return to Zero Inverted) format. This is a function of the relative voltage on the data line, representing the 'MARK' and 'SPACE' levels. This parameter determines whether a 'MARK' is represented by a high or low level.

Configure the Chameleon to use the same data encoding format as the (UUT).

**Bit Rate**

The bit rate determines the speed that the Chameleon 32 will transmit data. If the Chameleon is configured as the DTE, the bit rate is automatically set to RECEIVE. This means that the Chameleon will send data at the same rate as the data it receives, thus matching the speed set by the UUT.

If the Chameleon is configured as the DCE, it is responsible for setting the bit rate. The valid range is 50 - 64000 bits per second.

**Station Type:**
**Network/Subscriber**

This parameter determines if the Chameleon is acting as a subscriber (terminal) or as a network device. One device (either the UUT or the Chameleon) is designated the network device and the other device is designated the subscriber device.

This setting determines the value of the Command/Response (C/R) bit. The C/R bit is set to 1 for commands from and responses to the network side of the link. The C/R bit is set to 0 for commands from and responses to the subscriber side of the link.
If the UUT is acting as a network device, configure the Chameleon as a subscriber device. If the UUT is acting as a subscriber device, configure the Chameleon as a network device. This parameter can be changed under program control using the SET command so that you can test the ability of the UUT to ignore commands from like devices.

Frame Window:
1-7 Frames

The Frame Window determines the maximum number of sequentially numbered I-frames which the Chameleon can have outstanding (unacknowledged) at any given time.

This parameter can be overridden under program control using the SET WINDOW command.

T1:
1 - 255 Seconds

T1 is the T1 timer that is started by the station when it sends a command frame. If T1 expires before the station receives a response, it retransmits the message and restarts the timer. The maximum number of retransmissions is determined by the value of N2. Following an N2 re-transmission, the station will take an appropriate recovery action.

T1 is in units of seconds, in the range 1 - 255. This parameter can be overridden under program control using the SET T1 command.

N1:
1 - 512 Bytes

N1 defines the maximum number of bytes that can be in a transmitted or received frame.

When the Chameleon transmits or receives a frame from the line, it checks the value of N1. If the Chameleon receives a frame longer than N1, it automatically sends an FRMR. If you attempt to BUILD a frame which is longer than the value of N1, an error message is displayed.

N1 is within the range 2 to 512 bytes. This parameter can be overridden under program control using the SET N1 command.

N2: 1 - 255
Retransmissions

N2 defines the maximum number of frame re-transmissions that can occur after successive lapses of the T1 timer, before deciding that the receiving device is not going to respond.

N2 is within the range 1 - 255. This parameter can be overridden under program control using the SET N2 command.
SIMP/L HDLC MNEMONICS

Introduction

This section describes the default mnemonic table for SIMP/L HDLC (see figure below). For a general description of the use of mnemonics in simulation, refer to Section 3.2.

Entries

The SIMP/L HDLC mnemonic table has two columns:

- Mnemonic name (1-6 characters)
- Field width in bits

The default SIMP/L HDLC mnemonic table is shown below. You can also create and save additional mnemonic tables using the mnemonic commands described in Section 3.5.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>FIELD WIDTH (BITS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCGN</td>
<td>4</td>
</tr>
<tr>
<td>PK10</td>
<td>8</td>
</tr>
<tr>
<td>P(S)</td>
<td>3</td>
</tr>
<tr>
<td>P(R)</td>
<td>3</td>
</tr>
<tr>
<td>PAD1</td>
<td>1</td>
</tr>
<tr>
<td>MBIT</td>
<td>1</td>
</tr>
<tr>
<td>DBIT</td>
<td>1</td>
</tr>
<tr>
<td>QBIT</td>
<td>1</td>
</tr>
<tr>
<td>PAD2</td>
<td>2</td>
</tr>
<tr>
<td>GFI</td>
<td>4</td>
</tr>
<tr>
<td>LCN</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.4-4: SIMP/L HDLC Mnemonics
SIMP/L HDLC COMMAND INDEX

Introduction

This section lists the commands that are specific to SIMP/L HDLC by function. For more information about a command, refer to the page number indicated.

Remember that you can also use the SIMP/L commands listed in Section 5.3 and the Chameleon BASIC commands listed in Section 3.5.

TABLE 5.4-5: SIMP/L HDLC MISCELLANEOUS COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>5.4-10</td>
<td>S or D</td>
<td>Assigns values to HDLC timers and parameters.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.4-13</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>

TABLE 5.4-6: SIMP/L HDLC READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNKSTAT</td>
<td>5.4-8</td>
<td>S or D</td>
<td>Returns a value that indicates the status of the link.</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.4-12</td>
<td>S or D</td>
<td>Displays the values of HDLC timers and parameters.</td>
</tr>
</tbody>
</table>
**LNKSTAT**

**Description**

LNKSTAT is a read-only variable that returns a value between zero and seven indicating the status of the link.

The value returned by LNKSTAT indicates the link status as displayed in the figure below. These values are valid for HDLC whether the Chameleon is configured as a network or subscriber.

<table>
<thead>
<tr>
<th>LNKSTAT VALUE</th>
<th>LINK STATUS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link Disconnected</td>
<td>The DCE received a DISC and returned a UA. In this state, the DCE reacts to the receipt of a SABM and transmits a DM response in answer to a received DISC command.</td>
</tr>
<tr>
<td>1</td>
<td>Link Connection Requested</td>
<td>A SABM has been transmitted, but a UA has not yet been received.</td>
</tr>
<tr>
<td>2</td>
<td>Frame Rejected</td>
<td>A FRMR or CMDR has been transmitted to report an error condition not recoverable by retransmission of an identical frame.</td>
</tr>
<tr>
<td>3</td>
<td>Disconnect Requested</td>
<td>A DISC has been transmitted, requesting that operation be suspended.</td>
</tr>
<tr>
<td>4</td>
<td>Information Transfer</td>
<td>A SABM has been transmitted, and a UA received. The DCE will accept and transmit I-frames and S-frames.</td>
</tr>
<tr>
<td>5</td>
<td>Local Station Busy</td>
<td>The local station transmitted an RNR.</td>
</tr>
<tr>
<td>6</td>
<td>Remote Station Busy</td>
<td>The remote station transmitted an RNR.</td>
</tr>
<tr>
<td>7</td>
<td>Local and Remote Stations Busy</td>
<td>Both the local and the remote stations transmitted RNRs.</td>
</tr>
</tbody>
</table>

Table 5.4-7: Link Status Values

**Type**

Read-Only Variable

**Syntax**

(See examples below.)

**Abbreviation**

LN.
See Also

STATUS

Example 1

This example displays the current value of LNKSTAT.

PRINT LNKSTAT

Example 2

This example displays a message with the link status.

IF LNKSTAT = 5 PRINT "LOCAL STATION BUSY"
SET

Description

In SIMP/L HDLC, the SET command sets the values of the following variables:

- N1
- N2
- T1
- Network or Subscriber
- Window

Each of the variables is described below. Use the STATUS command to display the current values of the variables.

N1

N1 defines the maximum number of bytes in a frame to transmit or receive, within the range 2 to 512 bytes.

When the Chameleon transmits or receives a frame from the line, it checks the value of N1. If the Chameleon receives a frame longer than N1, it automatically sends an FRMR. If you attempt to BUILD a frame which is longer than the value of N1, an error message is displayed.

N2

N2 defines the maximum number of frame re-transmissions that can occur after successive lapses of the T1 timer, before deciding that the receiving device is not going to respond. N2 is within the range 1 - 255 and may not be set to 0.

T1

The T1 timer is started by the Chameleon when it sends a command frame. If T1 expires before the Chameleon receives a response, it retransmits the message and restarts the timer. The maximum number of retransmissions is determined by the value of N2. Following an N2 retransmission, the station will take an appropriate recovery action.

T1 is in units of seconds, in the range 1 - 255.

Network

You may choose to simulate a network or subscriber under program control, thereby overriding the selection made in the parameter set-up menu. When simulating a network, the Chameleon sends commands using the address 03 and sends responses using the address 01.

Subscriber

You may choose to simulate a network or subscriber under program control, thereby overriding the selection made in the parameter set-up menu. When simulating a subscriber, the Chameleon sends commands using the address 01 and sends responses using the address 03.
Chameleon BASIC Simulation Manual

Window
Window defines the maximum number of sequentially numbered I-frames which the DCE or DTE can have outstanding (unacknowledged) at any given time. Window must be in the range 1 - 7.

Type
Statement or Direct

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET N1 = x</td>
<td>x = 2 - 512 bytes</td>
</tr>
<tr>
<td>SET N2 = x</td>
<td>x = 1 - 255 bytes</td>
</tr>
<tr>
<td>SET T1 = x</td>
<td>x = 1 - 255 bytes</td>
</tr>
<tr>
<td>SET WINDOW = x</td>
<td>x = 1 - 7 bytes</td>
</tr>
<tr>
<td>SET NETWORK</td>
<td></td>
</tr>
<tr>
<td>SET SUBSCRIBER</td>
<td></td>
</tr>
</tbody>
</table>

Abbreviation
None

See Also
STATUS

Example 1
This example sets the HDLC parameters and then displays their status.

```
10 SET N1 = 128
20 SET N2 = 5
30 SET T1 = 50
40 SET WINDOW = 3
50 SET NETWORK
60 STATUS
!RUN
```

Result
Link-Disconnected
Timer T1 Value - 50
Maximum frame size (n1) - 128
Number of Re-transmissions (N2) - 5
Window - 3
Simulating a network.

Example 2
This program sets the value of N1 to 10 and then attempts to build a frame that exceeds 10 characters.

```
10 $A = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 SET N1 = 10
30 BUILD $A
!RUN
```

Result
Overflow!
STATUS

Description
The STATUS command displays the status of the link and the current values of the HDLC parameters and timers. These timers and parameters are described on the previous page (SET command).

Sub-states such as waiting acknowledgement or reject sent are also included in the SIMP/L software, but are not displayed by the command STATUS.

Type
Statement or Direct

Syntax
STATUS

Abbreviation
ST.

See Also
SET

Example
Result: Link Disconnected
Timer T1 Value - 200
Maximum frame size (N1) - 404
Number of Re-transmissions - 100
Window - 1
Simulating a network
**TPRINT**

**Description**

The TPRINT command displays the contents of the trace buffer, which stores the transmitted and received traffic (following a REC command). The format of the display is protocol-specific. Refer to the appropriate SIMP'L for a description of the TPRINT display for the protocol you are using.

You can stop the list using CTRL S. Resume the list with CTRL Q. Abort using the ESC key.

The trace buffer display is protocol-specific. The general format of the trace buffer display is as follows:

```
T = Transmitted frame  Time Stamp
R = Received frame     Hours:Minutes:Seconds: 01Seconds: 0001 Seconds
                  Protocol-specific
? = Bad CRC (frame overflow or abort)  First byte of frame
Space = Good CRC
```

**Type**

Statement or Direct

**Syntax**

TPRINT

**Abbreviation**

TP.

**See Also**

TLOAD, TSAVE, LTPRINT

**Example**
SAMPLE SIMP/L HDLC PROGRAMS

Introduction

These sample SIMP/L HDLC programs demonstrate several major concepts to consider when developing applications, but should not be considered valid tests for any particular application.

SHDLC1

You can use the transmission and reception commands described in the SIMP/L section to transmit, receive, and analyze the HDLC frame level.

The diagram below shows a sample X.25 packet, with the packet control header fields and their widths. Mnemonics will be used in the sample program to break a received frame into these fields.

<table>
<thead>
<tr>
<th>Octet</th>
<th>General Format Indicator (4 bits)</th>
<th>Logical Channel Group Number (4 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Logical Channel Number (8 bits)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Packet Type Identifier (8 bits)</td>
<td></td>
</tr>
</tbody>
</table>

Packet Control Header Detail

The sample program does the following:

- Defines the lengths of the mnemonics
- Assigns values to the mnemonics
- Assembles the mnemonics for transmission
- Transmits the mnemonics

The sample program receives a response, analyzes it, and displays a message that indicates the success of the test.

The following sample program demonstrates using these commands to send a RESTART packet and attempting to receive a RESTART CONFIRMATION.
Lines 10-40 define mnemonics for assembling and disassembling frames. The numbers indicates the field width in bits.

Establishes a link at level 2.

Lines 80-110 assign values to the mnemonics. PKID = Restart packet.

$C$C is the restart cause and diagnostic.

Defines the message in the transmission buffer.

Transmits the contents of the transmission buffer.

Waits 120 seconds for a response. Attempts to receive a frame.

Executes line 230 if frame received.

If timer has not expired and nothing was received, loop to receive again.

If timer elapsed and nothing was received, displays message.

Stops program execution after displaying message.

Executes remaining lines if something was received.

Disassembles received frame into indicated elements.

Tests Packet ID for Restart Confirmation packet.

260 PRINT "WRONG PACKET TYPE RECEIVED...ERROR CONDITION"

Tests packet 1D for Restart Confirmation packet.
SHDLC2  This program takes string A from the file named DATAFILE, builds it into a frame, and then transmits it.

```
10 CLEAR           Clears trace buffer.
20 OPEN "I", "DATAFILE"  Opens an input file.
30 READ $A        Reads a record and assigns it to $A.
40 BUILD $A       Assembles string A for transmission.
50 SLON           Sets the link level on.
60 TRAN           Transmits contents of transmission buffer.
70 TPRINT         Displays the trace buffer contents.
80 CLOSE          Closes the data file
90 STOP           End of program.
```

SHDLC3  This program receives string A and writes it into the file named DATAFILE.

```
20 REC                Receives.
30 BREAK $A           Disassembles received string.
40 WRITE $A           Writes $A to the file.
50 CLOSE              Closes the file.
```
SHDLC5

This program is a SIMP L HDLC call generator. It transmits, checks and receives X.25 packets.

```
200 DEFINE "COLEN" = 4
210 DEFINE "CGLEN" = 4
220 DEFINE "GFI" = 4
230 DEFINE "LCGM" = 4
231 DEFINE "PAD" = 1
232 DEFINE "LCN" = 8

Lines 200-232 define mnemonics. Numbers represent field width in bits.

240 PAD = 0
241 $C = CHR$(&1B) + "T"

$C will be used to position the cursor on the screen.

250 CLS
260 XYPLOT(3,30)
270 BLKREV

Clear the screen.
Position cursor at line 30, column 3.
Causes subsequent text to blink in reverse video.

280 ?"CALL TESTER V1.0"
290 NRM
300 ?
310 ?

Displays message.
Return to normal text display.
Print a blank line.
Print a blank line.

320 INPUT "ENTER GFI" GFI
340 INPUT "ENTER LCGN" LCGN
360 INPUT "ENTER STARTING LCN" S
380 INPUT "ENTER ENDING LCN" E
400 INPUT "ENTER LCN INCREMENT" I

Lines 320-400 prompt for mnemonic values and store result in numeric variables.

420 PRINT "MESSAGE TO TRANSMIT"
430 $INPUT $M
440 PRINT "CALLING ADDRESS"
450 $INPUT $G
460 PRINT "CALLED ADDRESS"
470 $INPUT $D

Lines 420-470 prompt for message and addresses and store result in string variables.
Stores length of calling address in numeric variable COLEN.
Stores length of called address in numeric variable CDLEN.
Display message while

480 CGLEN = LEN($G)
490 COLEN = LEN($D)

500 PRINT "PLEASE WAIT..."

510 REM Break down addresses for reassembly into BCD
550 $A = $G + $D
560 $A = BCD$(SA)
800 REM $A = addresses.
801 REM Prints headings.
802 GOSUB 10000
805 $A = $A + CHR$(0)
```

TEKELEC 5.4-17 6/11/90
Chameleon BASIC Simulation Manual

Ch. 5.4: SIMP/L-HDLC

830 FOR LCN = S TO E STEP I
831 XYPL0T(10,44)
832 ?\%LCN
835 CLEAR
840 PKID = &08

Set up loop to set the LCN according to the previously input values.
Position cursor at line 10 column 44.
Display hex value of current LCN.
Clear the trace buffer.
Assign value to packet ID mnemonic.

850 BUILD GFI, LCGN, LCN, PKID, CGLEN, CDLEN, SA

Build call packet.
Position cursor at line 12 column 44.
Display message
Normal video text display.
Transmit contents of buffer.
Execute subroutine that gets call confirmation.
Ready to transmit data.
Position cursor at line 12 column 44.
Display message.
Normal video text display.
Set P(S) value.
Set P(R) value.

ASSEMBLE DATA FOR MESSAGE.

920 BUILD GFI, LCGN, LCN, P(R), MBIT, P(S), PAD, SR

Build message in buffer.
Transmit contents of buffer.
Increment P(S).
Assemble data for second message.

950 BUILD GFI, LCGN, LCN, P(R), MBIT, P(S), PAD, SR

Build message in buffer.
Transmit contents of buffer.
Execute subroutine that gets RR.
Increment P(S).
Assemble data for second message.

1000 BUILD GFI, LCGN, LCN, P(R), MBIT, P(S), PAD, SR

Build message in buffer.
Transmit contents of buffer.
Packet ID = CLEAR.
Build CLEAR packet.
Transmit CLEAR packet.
Position cursor at line 12 column 44.
Display message.
Normal video text display.
Execute subroutine to get CLEAR.
1080 NEXT LCN
1083 XYPLOT(14,35)
1084 BLKREV
1085 ?"T E S T O K I"
1086 NRM
1087 STOP
1090 STOP
5010 GOSUB 9000

5020 IF PKID = &41 GOTO 5040
5021 REM If an old packet RR, have a window of 2.
5022 Z = PKID AND 1
5023 IF Z = 1 GOTO 5010
5030 ? "PACKET RR NOT RECEIVED"
5035 GOTO 5010
5040 RETURN
6000 REM Get Call Confirmation.
6001 REM First determine if there is an old packet RR outstanding.
6010 GOSUB 9000

6020 IF PKID = &OF GOTO 6030
6021 Z = PKID AND 1
6022 IF Z = 1 GOTO 6010
6025 ? "CALL CONF NOT RECEIVED"
6026 GOTO 6010
6030 RETURN
7010 GOSUB 9000

7020 IF PKID = &17 GOTO 7030
7021 REM Checks for old packet RR.
7022 Z = PKID AND 1
7023 IF Z = 1 GOTO 7010
7025 ? "CLEAR CONF NOT RECEIVED"
7026 GOTO 7010
7030 RETURN
9000 REM Receives something from the line with 3 minute timer
9020 TIM2 = 180
9030 IF TIM2 = 0 GOTO 9090
9035 REC
9040 IF LENGTH = 0 GOTO 9030
9050 BREAK GFI,LGN,LCN,PKID,$X
9060 RETURN
9090 ? " RECEIVER TIMED OUT"
9091 STOP
10000 CLS
10010 XYPLOT(1,30)
10020 BLKREV

CONFIRMATION.
Increment LCN loop counter and jump to top of loop (line 830).
Position cursor at line 34 column 35

Sets text to blink in reverse video.
Display message.
Sets normal text mode.
Stops program execution.

Execute subroutine to get RR2 packet.

Display message.
End of subroutine 5000.

Subroutine to get clear confirmation.
Displays message.
Ends subroutine.

If timer expires, displays message.
Stops program execution.
Clears the screen.
Position cursor at line 1, column 30.
Sets text to blink in reverse video.
10030 ? "CALL TESTER V1.0"
Displays message.
10040 NRM
Normal video text display.
10050 XYPLOT(2,5)
Position cursor at line 2, column 5.
10060 PRINT REV "GFI = "
Prints message in reverse video.
10080 PRINT NRM %GFI
Prints value of variable, in normal display.
10100 XYPLOT(2,20)
Position cursor at line 2, column 20.
10110 PRINT REV "LCGN = "
Prints message in reverse video.
10130 PRINT NRM %LCGN
Prints value of variable, in normal display.
10150 XYPLOT(2,40)
Position cursor at line 2, column 40.
10160 PRINT REV "LCN = "
Prints message in reverse video.
10180 PRINT NRM %LCN
Prints value of variable, in normal display.
10190 PRINT " "
Inserts two blank spaces.
10200 PRINT REV "TO"
Prints message in reverse video.
10220 PRINT NRM %E
Prints value of variable, in normal display.
10240 XYPLOT(3,5)
Position cursor at line 3, column 5.
10250 PRINT REV "MESSAGE = "
Prints message in reverse video.
10270 PRINT NRM $M
Prints string M. in normal text.
10290 XYPLOT(4,1)
Position cursor at line 4, column 1.
10300 FOR A = 1 TO 79
Set up loop to repeat 79 times.
10310 ? "="
Print equal sign.
10320 NEXT A
Increment loop counter A.
10330 ?
Position cursor at line 10, column 30.
10340 XYPLOT(10,30)
Print message in reverse video.
10350 ? "CURRENT LCN ="
Normal text display.
10360 PRINT REV %LCN
Position cursor at line 12, column 27.
10370 NRM
Print message.
10380 XYPLOT(12,27)
Print message in reverse video.
10390 PRINT "CURRENT STATUS ="
Normal text display.
10400 PRINTN REV "INITIALIZING"
Returns from GOSUB.
5.5 SIMP/L SDLC

General Characteristics

The general characteristics for SIMP/L SDLC are as follows:

- Point-to-point, half duplex
- Bit rate up to 64 Kbps with clocks received or supplied
- Maximum frame length 512 bytes
- Maximum number of outstanding frames is seven
- Line Code is NRZ or NRZI
- Simulation of DCE or DTE
- Simulation of Primary or Secondary stations
- CRC CCITT standards

Flow Control

Flow control occurs at the frame level, and depends on the amount of buffer space available on the Chameleon to receive data. If the receive buffer becomes full, the Chameleon declares the station busy and sends an RNR to the device under test. When the buffer is cleared, the Chameleon sends an RR to indicate that the busy condition has cleared.

To keep the receive buffer clear and avoid a station busy condition from disrupting your test, it is recommended that your program include a number of REC commands.
The format of an SDLC frame is shown below with the field widths in bits indicated.

<table>
<thead>
<tr>
<th>Field</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
<td>8</td>
</tr>
<tr>
<td>Address</td>
<td>8</td>
</tr>
<tr>
<td>Control Field</td>
<td>8 or 16</td>
</tr>
<tr>
<td>Information Field</td>
<td>Variable Length</td>
</tr>
<tr>
<td>Frame Check Sequence</td>
<td>16</td>
</tr>
<tr>
<td>Flag</td>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 5.5-1: SDLC Frame Format**

The opening and closing flags signal the end of the preceding frame and the beginning of the next frame, or is transmitted as fill time between multiple frames. A flag has the unique bit pattern: 0111 1110.

The **Address Field** contains the address of the station that is receiving a command or sending a response to a command.

The **Frame Check Sequence** (FCS) determines if the data in the packet was received without error. It is a 16-bit Cyclic Redundancy Check (CRC) that is calculated from the data.

The **Control Field** contains 8 or 16 bits and identifies the type of frame being transmitted. The following figure lists the frames for 8-bit control fields.
<table>
<thead>
<tr>
<th>Format</th>
<th>Commands</th>
<th>Responses</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>I (information)</td>
<td>0</td>
<td>N(S)</td>
</tr>
<tr>
<td>transfer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supervisory</td>
<td>RR (receive ready)</td>
<td>1 0 0 0</td>
<td>P/F</td>
</tr>
<tr>
<td></td>
<td>RNR (receive not ready)</td>
<td>1 0 1 0</td>
<td>P/F</td>
</tr>
<tr>
<td></td>
<td>REJ (reject)</td>
<td>1 0 0 1</td>
<td>P/F</td>
</tr>
<tr>
<td>Unnumbered</td>
<td>UI (unnumbered information)</td>
<td>1 1 0 0</td>
<td>P/F</td>
</tr>
<tr>
<td></td>
<td>NSI (non sequenced I-frame)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RIM/RQI (request initialization mode)</td>
<td>1 1 1 0</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>SIM (set initialization mode)</td>
<td>1 1 1 0</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>DM (disconnect mode)</td>
<td>1 1 1 1</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>UP (unnumbered poll)</td>
<td>1 1 0 0</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>RD (request disconnect)</td>
<td>1 1 0 0</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>DISC (disconnect)</td>
<td>1 1 0 0</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>UA (unnumbered acknowledge)</td>
<td>1 1 0 0</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>SNRM (set normal response mode)</td>
<td>1 1 0 0</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>FRMR (frame reject)</td>
<td>1 1 1 0</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>XID (exchange identification)</td>
<td>1 1 1 1</td>
<td>P/F</td>
</tr>
<tr>
<td></td>
<td>XID (exchange identification)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CFGR (configure)</td>
<td>1 1 1 0</td>
<td>P/F</td>
</tr>
<tr>
<td></td>
<td>TEST (test)</td>
<td>1 1 0 0</td>
<td>P/F</td>
</tr>
<tr>
<td></td>
<td>BCN (beacon)</td>
<td>1 1 1 1</td>
<td>F</td>
</tr>
</tbody>
</table>

Figure 5.5-2: HDLC Frame Formats (8-bit Control Field)
**SIMP/L SDLC PARAMETER SET-UP MENU**

**Introduction**

This section describes the parameters and valid values for the SIMP/L SDLC menu. This menu enables you to configure and save parameters for running SIMP/L SDLC simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu (option S), refer to Section 2.3.

---

**Menu Options**

The choices for the SIMP/L SDLC parameters are listed below. Each parameter is described on the next page.

A. Simulate
   - DCE
   - DTE

B. Data Encoding
   - NRZ
   - NRZI

C. Bit Rate
   - Range: 50 - 64000 bits (DCE)
   - Received (DTE)

D. Station Type
   - Primary
   - Secondary

E. Station Address
   - Range: 00-FF hex

F. T1
   - Range: 1 - 255 seconds

G. T2
   - Range: 1 - 255 seconds

H. N2
   - Range: 0 - 99 re-transmissions

---

**Figure 5.5-3 : SIMP/L SDLC Parameter Set-Up Menu**
If you are unfamiliar with these parameters, use the following guidelines for assistance in selecting the correct settings.

**Simulate:**
**DCE/DTE**

The DCE furnishes the clock which determines the data rate. It also determines on which connector pins the data is physically sent and received. For Simulation, one device is designated as the DCE; the other device is designated as a DTE.

If the Unit Under Test (UUT) is supplying the clock, it is the DCE. In this case, the Chameleon must be configured as a DTE. If the UUT is not supplying the clock, it is designated as a DTE. The Chameleon must then be configured as the DCE.

**Data Encoding:**
**NRZ/NRZI**

This parameter determines if the data is transmitted in NRZ (Non-Return to Zero) or NRZI (Non-Return to Zero Inverted) format. This is a function of the relative voltage on the data line, representing the 'MARK' and 'SPACE' levels. This parameter determines whether a 'MARK' is represented by a high or low level.

**Bit Rate**

The bit rate determines the speed that the Chameleon 32 will transmit data. If the Chameleon is configured as the DTE, the bit rate is automatically set to RECEIVE. This means that the Chameleon will send data at the same rate as the data it receives, thus matching the speed set by the UUT.

If the Chameleon is configured as the DCE, it is responsible for setting the bit rate. The valid range is 50 - 64000 bits per second.

Configure the Chameleon to use the same data encoding for

**Station Type:**
**Primary/Secondary**

In SDLC, one side of the link is the Primary, or commanding station. The other side of the link is referred to as the Secondary (responding) station. If the UUT is the Primary station, configure the Chameleon as the Secondary. If the UUT is the Secondary station, configure the Chameleon as the Primary station.
Station Address: 00-FF HEX

ADDR is the station address of either the Secondary station. Any frame that is received with a different address is discarded. The address must be within the range of 00 to FF hexadecimal. This parameter can be overridden under program control using the SET ADDR command.

T1:
1 - 255 Seconds

T1 is the T1 timer that is started by the station when it sends a command frame. If T1 expires before the station receives a response, it retransmits the message and restarts the timer. The maximum number of retransmissions is determined by the value of N2. Following an N2 re-transmission, the station will take an appropriate recovery action.

T1 is in seconds, in the range 1 - 255. This parameter can be overridden under program control using the SET T1 command.

T2:
1-255 Seconds

T2 defines the amount of time that can elapse between sending polled frames. It is used by the primary to poll the secondary in the absence of traffic, usually by sending a polled RR, or if the local station is busy, a polled RNR.

T2 is in seconds, with a valid range from 1 to 255, inclusive. This parameter can be overridden under program control using the SET T2 command.

N2: 1 - 255 Retransmissions

N2 defines the maximum number of frame re-transmissions that can occur after successive lapses of the T1 timer, before deciding that the receiving device is not going to respond.

N2 is within the range 0 - 99 retransmissions. This parameter can be overridden under program control using the SET N2 command.
SIMP/L SDLC MNEMONICS

There is no default mnemonic table for SIMP/L SDLC, but you can define and save your own mnemonic tables using the mnemonic commands described in Section 3.5.

Refer to Section 3.2 for a general description of the use of mnemonic tables in simulation.
SIMP/L SDLC COMMAND INDEX

Introduction This section lists the commands by function that are specific to SIMP/L SDLC. For more information about a command, refer to the page number indicated.

Remember that you can also use the SIMP/L commands listed in Section 5.3 and the Chameleon BASIC commands listed in Section 3.5.

### TABLE 5.5-4: SIMP/L SDLC READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNKSTAT</td>
<td>5.5-9</td>
<td>Variable</td>
<td>Returns a value that indicates the status of the link between the primary and secondary stations.</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.5-12</td>
<td>S or D</td>
<td>Displays the status of the link and the values of SDLC timers and parameters.</td>
</tr>
</tbody>
</table>

### TABLE 5.5-5: SIMP/L SDLC TRANSMISSION AND RECEPTION COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSI</td>
<td>5.5-10</td>
<td>S or D</td>
<td>Transmits an NSI frame assembled using the BUILD command.</td>
</tr>
<tr>
<td>TEST</td>
<td>5.5-13</td>
<td>S or D</td>
<td>Transmits a test frame assembled using the BUILD command.</td>
</tr>
<tr>
<td>XID</td>
<td>5.5-15</td>
<td>S or D</td>
<td>Transmits an XID frame defined using the XIDFLD command.</td>
</tr>
<tr>
<td>XIDFLD</td>
<td>5.5-16</td>
<td>S or D</td>
<td>Sets the data field of an XID frame.</td>
</tr>
</tbody>
</table>

### TABLE 5.5-6: SIMP/L SDLC MISCELLANEOUS COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>5.5-11</td>
<td>S or D</td>
<td>Assigns values to SDLC timer and parameters.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.5-14</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
LNKSTAT

Description

LNKSTAT is a read-only variable that displays the status of the link between the primary and the secondary stations. The value returned by this variable ranges from 0 to 6, as shown in the tables below.

<table>
<thead>
<tr>
<th>PRIMARY</th>
<th>LINK STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal Disconnected Mode</td>
</tr>
<tr>
<td>1</td>
<td>Link Request State</td>
</tr>
<tr>
<td>2</td>
<td>Disconnect Request State</td>
</tr>
<tr>
<td>3</td>
<td>Information Transfer State</td>
</tr>
<tr>
<td>4</td>
<td>Local Station Busy</td>
</tr>
<tr>
<td>5</td>
<td>Remote Station Busy</td>
</tr>
<tr>
<td>6</td>
<td>Local and Remote Stations Busy</td>
</tr>
</tbody>
</table>

Table 5.4-7: Chameleon Configured as Primary

<table>
<thead>
<tr>
<th>SECONDARY</th>
<th>LINK STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal Disconnected Mode</td>
</tr>
<tr>
<td>1</td>
<td>Initialization Mode</td>
</tr>
<tr>
<td>2</td>
<td>Frame Reject Mode</td>
</tr>
<tr>
<td>3</td>
<td>Information Transfer State</td>
</tr>
<tr>
<td>4</td>
<td>Local Station Busy</td>
</tr>
<tr>
<td>5</td>
<td>Remote Station Busy</td>
</tr>
<tr>
<td>6</td>
<td>Local and Remote Stations Busy</td>
</tr>
</tbody>
</table>

Table 5.4-8: Chameleon Configured as Secondary

Type

Read-Only Variable

Syntax

(See examples below.)

Abbreviation

LN.

See Also

STATUS, SLON, SLOF

Example 1

This example displays the value of LNKSTAT.

PRINT LNKSTAT

Example 2

This example displays a link status message.

IF LNKSTAT = 4 PRINT "LOCAL STATION BUSY"
NSI

Description
The NSI command transmits an NSI frame (non-sequenced I-frame) that was assembled using the BUILD command. Refer to Table 5.4-1 for a description of an NSI frame.

Type
Statement or Direct

Syntax
NSI

Abbreviation
NS.

See Also
BUILD, TPRINT

Example
10 $A = "HELLO"
20 BUILD $A
30 NSI
40 TPRINT
!RUN
## SET

**Description**

In SIMP/L SDLC, the SET command sets the value of the following SIMP/L SDLC variables: T1, T2, N2, station address.

Each variable is described below.

<table>
<thead>
<tr>
<th>ADDR</th>
<th>ADDR is the station address of either the primary or secondary station. Any frame that is received with a different address is discarded. The addresses must be within the range of 0 to FF hexadecimal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>N2 defines the maximum number of frame re-transmissions that can occur after successive lapses of the T1 timer, before deciding that the receiving device is not going to respond. It is used by the primary station only. The range of N2 is 1 to 99 tries.</td>
</tr>
<tr>
<td>T1</td>
<td>The T1 timer is started by the station when it sends a command frame. If T1 expires before the station receives a response, it retransmits the message and restarts the timer. The maximum number of retransmissions is determined by the value of N2. Following N2 re-transmissions, the station will take an appropriate recovery action. T1 is in units of tenths of seconds, with a valid range from 1 and 255, inclusive. T1 cannot be set to zero.</td>
</tr>
<tr>
<td>T2</td>
<td>T2 defines the amount of time that can elapse between sending polled frames. It is used by the primary to poll the secondary in the absence of traffic, usually by sending a polled RR, or if the local station is busy, a polled RNR. T2 is in units of tenths of seconds, with a valid range from 1 to 255, inclusive.</td>
</tr>
</tbody>
</table>

**Type**

Statement or Direct

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET ADDR = x</td>
<td>x is in the range 00 - FF hex</td>
</tr>
<tr>
<td>SET N2 = x</td>
<td>x is in the range 1 - 99</td>
</tr>
<tr>
<td>SET T1 = x</td>
<td>x is in the range 1 - 255</td>
</tr>
<tr>
<td>SET T2 = x</td>
<td>x is in the range 1 - 255</td>
</tr>
</tbody>
</table>

*where:* x is a valid numeric value or variable.

**Example**

```
SET N2 = 10
SET T2 = 50
SET T1 = 50
SET ADDR = 96
```
## STATUS

<table>
<thead>
<tr>
<th>Description</th>
<th>The STATUS command gives the status of the link and the values of the SIMP/L SDLC timers and parameters. These timers and parameters are described on the previous page (SET command).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>STATUS</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>STA.</td>
</tr>
<tr>
<td>See Also</td>
<td>SET</td>
</tr>
<tr>
<td>Example</td>
<td>ISTATUS</td>
</tr>
</tbody>
</table>

Normal Disconnect Mode (NDM)

Timer T1 Value - 10
Timer T2 Value - 10
Number of Re-transmissions - 10
Station address - 96
**TEST**

**Description**
The TEST command sends a test frame that was assembled using the BUILD command. Refer to Table 5.4-1 for a description of the TEST frame format.

**Type**
Statement or Direct

**Syntax**
TEST

**Abbreviation**
TE.

**See Also**
TRAN, BUILD, NSI

**Example**
```
10 $A = "HELLO"
20 BUILD $A
30 TEST
40 TPRINT
```
### TPRINT

**Description**
The TPRINT command displays the contents of the trace buffer, which contains the received and transmitted traffic.

The format of the trace buffer display for SIMP/L SDLC is shown below. For more information about the use of the trace buffer in simulation, refer to Section 3.2.

**Type**
Statement or Direct

**Syntax**
TPRINT

**Abbreviation**
TP.

**See Also**
TSAVE, TLOAD, REC

**Example**
TPRINT
XID

Description
The XID command transmits an XID frame defined using the XIDFLD command. It is only valid when the Chameleon is configured as a primary station.

In the interest of high-speed operation, the Chameleon ignores certain frame types such as XID. Therefore, a frame transmitted using XID will not appear in the trace buffer.

Refer to Table 5.4-1 for a description of the XID frame format.

Type
Statement or Direct

Syntax
XID

Abbreviation
None

See Also
TRAN, NSI, TEST

Example
This example assigns an XID data field and then transmits the frame.

10 $A = "123456"
20 XIDFLD=$A
30 XID
XIDFLD

Description
The XIDFLD command sets the data field of an XID frame.

Type
Statement or Direct

Syntax
XIDFLD = $A

where: $A is the data field with a required length of 6 bytes. If you attempt to set an XID field with a string less than 6 bytes long, the following message appears:

String not long enough. XID field must be six bytes.

If you attempt to set the XID field with a string that is longer, than 6 bytes, only the first six bytes of the string are used.

Abbreviation
XI.

See Also
XID

Example
This example assigns an XID data field and then transmits the frame.

10 $A = "123456"
20 XIDFLD=$A
30 XID
SIMP/L SDLC SAMPLE PROGRAMS

The following examples of SIMP/L SDLC programs are for demonstration only. They demonstrate major concepts to consider when developing applications, but should not be considered valid tests for any particular application.

SSPLC1

You can use the SIMP/L transmission and reception commands described in Section 2 to transmit, receive, and analyze the SDLC frame level. This program demonstrates how an SNA 1-field can be transmitted, received, and analyzed.

The figure below illustrates an SNA frame with a FID type 2 transmission header. Each mnemonic is defined with the field width in bits indicated.

![Figure 5.5-9: SNA FID type 2 Transmission Header](image)

---

**TEKELEC**

5.5-17

6/11/90
Lines 10-60 defines mnemonics for the transmission header fields.

Assigns values to mnemonic. The ampersand symbol (&) assigns hexadecimal values.

Assembles transmission header in buffer.

Sets the link on.

Transmits the contents of the transmission buffer.

Displays the block in hexadecimal. The result is 24 1E 20 1B 00.

To demonstrate analyzing data, the program receives the same block that was transmitted above and uses the BREAK command to disassemble it.

Receives the block.

If nothing received, tries again.

Disassembles the received frame using mnemonics.

Displays last received block on screen.

Displays mnemonics of received block.
SS DLC2

This program establishes a link, and transmits three I-frames on addresses C0 through C8.

5 CLEAR
10 GOSUB 2000
20 FOR A = &C0 TO &C8

30 GOSUB 1000,
40 NEXT A
50 STOP

Clears trace buffer.
Sets string values for $A, $B, $C.
Loops to send messages on each channel.
Executes subroutine.
Increments A from C0 to C8 hex.
Terminates program.

100 REM SUBROUTINE BUILDS & TRANSMITS STRINGS

1000 SET ADDR = A
1010 SLOW Sets link on.
1020 BUILD $A

Sets address to loop index value.
Assembles and transmits first message.

1030 TRAN
1035 TPRINT
1040 BUILD $B

Displays trace buffer contents.
Assemble and transmits second message.

1050 TRAN
1055 TPRINT
1060 BUILD $C

Displays trace contents.
Assembles and transmits third message.

1070 TRAN
1075 TPRINT
1080 SLOF
1090 RETURN

Displays trace contentS.
Disconnects link.
End of subroutine line 1000.

1100 REM THIS SUBROUTINE DEFINES STRINGS

2000 $A = "MESSAGE NO. 1" + CHR$(&D)+CHR$(&A)
2010 $B = "MESSAGE NO. 2" + CHR$(&D)+CHR$(&A)
2020 $C = "MESSAGE NO. 3" + CHR$(&D)+CHR$(&A)

2030 RETURN End of subroutine line 2000
SSDLC3

This program takes $A$ from the file named DATAFILE, builds it into a frame, and then transmits it.

10 CLEAR
   Clears trace buffer.
20 OPEN "I","DATAFILE"
   Opens a file for input.
30 READ $A$
   Reads a record and assigns it to $A$.
40 BUILD $A$
   Assembles $A$ for transmission.
45 SLOM
   Sets the link on.
50 TRAN
   Transmits.
55 SLOF
   Sets the link off.
60 TPRINT
   Displays trace buffer contents.
70 CLOSE
   Closes file opened in line 10.
80 STOP
   Terminates execution.

SSDLC4

This program opens an output file, receives $A$, and writes it into the file named DATAFILE.

10 CLEAR
   Clears trace buffer.
15 SLOM
   Set the link on.
20 OPEN "O","DATAFILE"
   Opens an output file.
30 REC
   Receive.
40 TPRINT
   Displays contents of trace buffer.
50 BREAK $A$
   Disassembles string $A$.
60 WRITE $A$
   Writes $A$ to the file.
70 IF RXLENGTH=0 GOTO 70
   End of file, so closes data file.
80 CLOSE
   Closes data file.
85 SLOF
   Sets link off.
90 STOP

In this program, the Chameleon simulates an SDLC 3274 with one terminal attached (FID 2). The Chameleon may simulate the DTE alone (in which case one line and two modems must be available) or the DTE and DCE simultaneously.

The Chameleon is the secondary station, with a 3705 being the primary station. The host computer access method (in this case, ACF/VTAM) activates the simulated 3274 and terminal. After the simulated terminal notifies the host of its existence, the access method will send a welcome message to the terminal.

5 REM *****AUTHOR ELIZABETH WALLING: WRITTEN JULY 1983***
10 L = 0
20 TIM2 = 60
100 REC
105 IF TIM2 = 0 GOTO 900
110 IF LENGTH = 0 GOTO 100

112 REM *****BREAK DOWN RECEIVED FRAME ********
113 REM A MNEM. TABLE MUST BE WRITTEN FOR THE FOLLOWING
115 BREAK FID,MFP,EFI,PAD2,DAF,OAF,SMF,RH,RUC,PAD,PA5,SDI,
BCl,PAD2,PAD2,$R
$R is the rest of the frame.
118 REM *****IF FIRST BIT OF RH = 1 *****
120 IF RH = 1 GOTO 20

140 REM *****OTHERWISE IT'S A REQUEST--WE MUST RESPOND *****
150 IF RUC = 0 GOTO 350
160 IF RUC = 3 GOTO 180
170 GOTO 950
180 $D = LEFTS($R,1)
190 $G = CHR$(80D)
200 IF $D =$G GOTO 280
210 $G = CHR$(111)
220 IF $D =$G GOTO 250
230 GOTO 990

250 REM *****RESPOND TO ACTPU ************
255 RH = 1

It's a response, so go back.

Check RU category, if 0 it's a message. If it isn't 0 or 3, displays the category and stop.

Gets first byte of RU. 3705 must activate devices. Is it hex 0D - ACTLU?

Is it hex 11 - ACTPU?

No. Display it and stop.

Set sbit 0 of RII to 1, indicates a response.
256 REM ******BUILD STRINGS FOR TRANSMISSION ******
257 $G = CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
260 $E = " "
265 $E=$D+CHR$(&11)+EBCS(SE)+CHR$(0)+CHR$(7)+CHR$(1)+$G
266 REM **RU IN HEX = 11114040404040404040000000000000000000
270 GOSUB 800
275 GOTO 20

Go to transmit routine.
Then get another frame.

280 REM ******RESPOND TO ACTLU *****
281 RH = 1
Sets bit 0 of RH to 1, indicates a response.

282 REM ******BUILD STRING FOR TRANSMISSION ****
285 $G = CHR$(0) + CHR$(0) + CHR$(0)
290 $H = CHR$(1)
295 $E=$D+$H+$SH+CHR$(0)+CHR$(&85)+$G+CHR$(&OC)+CHR$(6)+$H+CHR$(0)+$H+$G
296 REM **RH IN HEX = 0D01010085000000C0601000100000
300 GOSUB 800
305 IF L = 1 GOTO 20
Go to transmit routine.
Get another frame if L = 1.

306 REM ****** IF L IS STILL = 0 THEN ******
307 REM **LET FIRST TERMINAL ACTIVATE AND SEND NOTIFY **
308 REM ******SO IT CAN RECEIVE THE WELCOME MESSAGE ******
310 EFI = 0
315 SNF = 0
320 RQH = &0B
First byte of RII is hex 0B.
325 $E=CHR$(&81)+CHR$(&20)+CHR$(&20)+CHR$(&OC)+CHR$(6)+CHR$(3)+CHR$(0)+$H+$G
327 REM **RU IN HEX = 8106200C06030001000000000000000
329 REM *****BUILD THE WHOLE PIU (FRAME) ****
330 BUILD FID,MPF,EFI,PADZ,OAF,DAF,SNF,RQH,PAD2,PAD2,$E
331 REM ***** NOTE: REMEMBER TO SWAP DAF AND OAF *****
335 TRAN
340 L = L + 1 Transmits it.
345 GOTO 20
Increments L to show that notify was sent.
Gets another frame.
350 REM ** WE COULD CHECK TO SEE WHAT MESSAGE THIS IS ****
351 REM* * BUT WE'LL JUST ASSUME IT'S THE WELCOME MESSAGE **
355 PRINT "THIS MUST BE THE WELCOME MESSAGE"
360 RDISPF

Display it to make sure.

365 STOP

800 REM * SUBROUTINE TO BUILD AND TRANSMIT A PIU (FRAME) **
805 BUILD FID, MPF, EFI, PAD2, OAF, DAF, SWF, RH, RUC, PAD, PAD, SDI, BCI, PAD2, PAD2, $E
810 TRAN

Transmit sit.

820 RETURN
900 PRINT "TIMED OUT"

End of subroutine.

905 STOP
950 PRINT "RUC = ", %RUC
955 STOP

Time out if line idle for 60 seconds.

980 PRINT "THE RU RECEIVED = " %SR
990 RDISPF

Print the RU category.

991 RDISPF
995 STOP

Display the whole PIU.
The user-defined mnemonics for the program above are listed in the table below.

<table>
<thead>
<tr>
<th>NAME</th>
<th>NO. OF BITS</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>4</td>
<td>Format Identifier</td>
</tr>
<tr>
<td>MPF</td>
<td>2</td>
<td>Mapping Field (middle, last, first or segment only)</td>
</tr>
<tr>
<td>EFI</td>
<td>2</td>
<td>Expedited Flow Indicator</td>
</tr>
<tr>
<td>PAD2</td>
<td>8</td>
<td>Rest of TH or RH not needed for breakdown</td>
</tr>
<tr>
<td>DAF</td>
<td>8</td>
<td>Destination Address Field</td>
</tr>
<tr>
<td>OAF</td>
<td>8</td>
<td>Origin Address Field</td>
</tr>
<tr>
<td>SNF</td>
<td>16</td>
<td>Sequence Number Field</td>
</tr>
<tr>
<td>RH</td>
<td>1</td>
<td>First bit of RH byte. Zero indicates requestor response.</td>
</tr>
<tr>
<td>RUC</td>
<td>2</td>
<td>RU category (part of the RH)</td>
</tr>
<tr>
<td>PAD</td>
<td>1</td>
<td>Fields not needed for breakdown.</td>
</tr>
<tr>
<td>SDI</td>
<td>1</td>
<td>Sense Data Indicator</td>
</tr>
<tr>
<td>BCI</td>
<td>2</td>
<td>Begin and End Chain Indicators (BCI + ECI)</td>
</tr>
<tr>
<td>RHQ</td>
<td>8</td>
<td>Byte 0 of the RH.</td>
</tr>
</tbody>
</table>

Figure 5.5-10: Sample Program Mnemonics
5.6 SIMP/L LAPD

Introduction

This section includes commands, variables, and sample programs for SIMP/L LAPD. You can use SIMP/L LAPD to emulate any protocol that uses a LAPD link level and to configure the LAPD link parameters.

Important!

There are two versions of the SIMP/L LAPD software on your simulator diskette. These two versions are:

- SIMP/L LAPD
- Extendable SIMP/L LAPD

The Extendable SIMP/L LAPD software requires the extended RAM that became available with newer Chameleons. When you load SIMP/L LAPD, the software checks for the extended RAM and automatically loads the version that is appropriate for your Chameleon. If you do not have extended RAM, the regular version of SIMP/L LAPD is loaded, and this is the only version you can use.

If you have extended memory, Extendable SIMP/L LAPD is loaded. However, the parameter set-up menu has an option which enables you to select between the extended and non-extended mode. Additionally, the EXTEND and UNEXTEND commands available in Extendable SIMP/L LAPD enable you to select extended or non-extended mode.

Extendable SIMP/L LAPD has features that are not available in the other version. These features are marked with asterisks (*) in this section.

To determine which version of SIMP/L LAPD start the SIMP/L LAPD simulator, as described in Section 2.1. If you do not have extended memory SIMP/L LAPD software is loaded and the screen displays SIMPL LAPD Version x.xx.

If you have the extended memory, Extendable SIMP/L LAPD software is loaded and the screen displays Extendable SIMPL LAPD Version x.xx.

If you want to upgrade your Chameleon to include extended memory, please call TEKELEC Customer Support.
General Characteristics

The general characteristics of SIMP/L LAPD are as follows:

- Full duplex
- Bit rate up to 64 Kbps with clocks received or supplied
- Maximum frame length of 512 bytes
- Maximum frame window of seven
- NRZ/NRZI encoding
- Simulation of DCE or DTE
- CRC CCITT Standard
- Frame level conforming to CCITT Q.921 (May 1984)
- Single SAPI (SIMP/L LAPD) or 3 SAPIs (Extendable SIMP/L LAPD)
- Dual TEIs (SIMP/L LAPD) or 4 TEIs (Extendable SIMP/L LAPD)

Control Features

The Extendable SIMP/L LAPD package enables you to configure a control status byte that enable you to:

- Poll SABM(E) and DISC frames on the first transmission
- Monitor the physical link at any time using XID frames
- Control XID frame polling (all polled, none polled, polled with I-fields, polled without I-fields)
- Respond to received SABM(E)s with a UA only or with a UA and a SABM(E)
- Transmit XID frames by program control with or without I-fields.
- Accept response frames matching any combination of user-defined receive SAPIs and TEIs, or restrict response frames accepted to those matching transmit values.

Two Extendable SIMP/L LAPD commands enable you to configure the status control byte:

- SET {fnctn}
  This command enables you to select individual control options using pre-defined mnemonics. For example, the command SET SBMCOL selects the control option that generates SABM(E) collisions.

- SET CONFIG
  This command enables you to set all control options with a single command by inserting a hex value into a bit-mapped configuration byte.

Refer to the SET {fnctn} and SET CONFIG commands in this section for more information.
Multiple Addressing

In Extendable SIMP/L LAPD, three user-defined SAPIs and TEIs can be active at any given time. Frames received with a SAPI and TEI matching any of the user-defined values, or a user-defined SAPI and a broadcast TEI (127), are accepted. Any response automatically has a matching SAPI and TEI.

To use fewer than three SAPIs and/or TEIs, you disable those you do not want to use. To disable a SAPI or TEI:

- Set the unused SAPI or TEI value to an invalid value, for example, a SAPI = 128.
- Set the unused SAPI or TEI value to the same value as a SAPI or TEI in use.

Refer to the SET RSAPI and SET RTEI commands for information about setting user-defined values. Refer to page 5.6-34 for more information about using multiple addressing.

Frame Status Byte

In Extendable SIMP/L LAPD, a frame status byte at the beginning of each received packet gives status information on frames received by the level 2 processor. It provides the following:

- Frame type
- SAPI and TEI that were matched
- Command or response frame
- Poll/Final bit value

If you are using Extendable SIMP/L LAPD, this byte is included in the trace buffer, which is displayed using the TPRINT command. You can also access the frame status byte using the FRSTAT variable. Refer to these commands for more information.
XID Frames

XID command and response frames can contain information fields with content that varies widely depending on the application in use. In Extendable SIMP/L LAPD, the transmit buffer is appended as an information field to any XID frame so that you can transmit information in any format, in both XID command and response frames. This is true except for those XID frames automatically transmitted as a physical link monitor when configured for XID exchange.

Note:

Many implementations of LAPD use XID command and response frames to exchange setup parameters, and then exchange XID frames without I-fields to monitor the condition of the line.

Transmission of an XID frame with an I-field does not alter the contents of the data buffer. Therefore, you must clear the buffer before exchanging non-data XID frames by program or manual control. Two ways to clear the buffer are:

1. Build a null string.

   500 $A="" 
   510 BUILD $A

   Clears string $A to null value.
   Clears transmit data buffer.

2. Enter the BUFFER command without a value.

   BUFFER = <RETURN>

   Refer to the TRXIDC and TRXIDR commands in this section for information about transmitting XID frames.

Flow Control

Flow control occurs at the frame level, and depends on the amount of buffer space available on the Chameleon to receive data. If the receive buffer becomes full, the Chameleon declares the station busy and sends an RNR to the device under test. When the buffer is cleared, the Chameleon sends an RR to indicate that the busy condition has cleared.

To keep the receive buffer clear and avoid a station busy condition from disrupting your test, it is recommended that your program include a number of REC commands.
Frame Format

The figure below illustrates the format of an LAPD frame with the field widths in bits indicated.

<table>
<thead>
<tr>
<th>FLAG</th>
<th>ADDRESS</th>
<th>CONTROL FIELD</th>
<th>I-FIELD</th>
<th>FRAMES CHECK SEQUENCE</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8 or 16</td>
<td>8 or 16</td>
<td></td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 5.6-1: Frame Format

The opening and closing flags signal the end of the preceding frame and the beginning of the next frame, or can be transmitted as fill time between multiple frames. A flag has the unique bit pattern: 0111 1110.

The Address Field contains the address of the station that is receiving a command or sending a response to a command.

The Frame Check Sequence (FCS) determines if the data in the packet was received without error. It is a 16-bit Cyclic Redundancy Check (CRC) that is calculated from the data.

The Control Field contains 8 bits which identify the type of frame being transmitted, as indicated in the tables on the following pages.
<table>
<thead>
<tr>
<th>Format</th>
<th>Commands</th>
<th>Responses</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information transfer</td>
<td>I (information)</td>
<td>N(R)</td>
<td>8 7 6 5 4 3 2 1 0 4</td>
</tr>
<tr>
<td>Supervisory</td>
<td>RR (receive ready)</td>
<td>N(R)</td>
<td>P/F 0 0 0 0 4</td>
</tr>
<tr>
<td></td>
<td>RR (receive not ready)</td>
<td>N(R)</td>
<td>P/F 0 1 0 1 4</td>
</tr>
<tr>
<td></td>
<td>REJ (reject)</td>
<td>N(R)</td>
<td>P/F 1 0 0 1 4</td>
</tr>
<tr>
<td>Unnumbered</td>
<td>SABM (set asynchronous balance mode)</td>
<td>0 0 1 P</td>
<td>1 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>DM (disconnect mode)</td>
<td>0 0 0 F</td>
<td>1 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>*SIO (sequenced information 0)</td>
<td>0 1 1 P/F</td>
<td>0 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>*SIO (sequenced information 1)</td>
<td>1 1 1 P/F</td>
<td>0 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>UI (unnumbered information)</td>
<td>0 0 0 P</td>
<td>0 0 1 1 4</td>
</tr>
<tr>
<td></td>
<td>DISC (disconnect)</td>
<td>0 1 0 P</td>
<td>0 0 1 1 4</td>
</tr>
<tr>
<td></td>
<td>UA (unnumbered acknowledge)</td>
<td>0 1 1 F</td>
<td>0 0 1 1 4</td>
</tr>
<tr>
<td></td>
<td>FRMR (frame reject)</td>
<td>1 0 0 F</td>
<td>0 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>XID (exchange identification)</td>
<td>1 0 1 P/F</td>
<td>1 1 1 1 4</td>
</tr>
</tbody>
</table>

Figure 5.6-2: SIMP/L LAP/D Frame Specifications (MOD8)
<table>
<thead>
<tr>
<th>Format</th>
<th>Commands</th>
<th>Responses</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information transfer</td>
<td>I (information)</td>
<td></td>
<td>8 7 6 5 4 3 2 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N(S)</td>
<td>0 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N(R)</td>
<td>P 5</td>
</tr>
<tr>
<td>Supervisory</td>
<td>RR (receive ready)</td>
<td>RR (receive ready)</td>
<td>0 0 0 0 0 0 0 1 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N(R)</td>
<td>p/f 5</td>
</tr>
<tr>
<td></td>
<td>RNR (receive not ready)</td>
<td>RNR (receive not ready)</td>
<td>0 0 0 0 0 1 0 1 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N(R)</td>
<td>p/f 5</td>
</tr>
<tr>
<td></td>
<td>REJ (reject)</td>
<td>REJ (reject)</td>
<td>0 0 0 0 1 0 0 1 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N(R)</td>
<td>p/f 5</td>
</tr>
<tr>
<td>Unnumbered</td>
<td>SABME (set asynchronous balance mode extended)</td>
<td></td>
<td>0 1 1 p 1 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DM (disconnect mode)</td>
<td>0 0 0 f 1 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>DISC (disconnect)</td>
<td></td>
<td>0 1 0 p 0 0 1 1 4</td>
</tr>
<tr>
<td></td>
<td>UA (unnumbered acknowledge)</td>
<td></td>
<td>0 1 1 f 0 0 1 1 4</td>
</tr>
<tr>
<td></td>
<td>FRMR (frame reject)</td>
<td></td>
<td>1 0 0 f 0 1 1 1 4</td>
</tr>
<tr>
<td></td>
<td>XID (exchange identification)</td>
<td>XID (exchange identification)</td>
<td>1 0 1 p/f 1 1 1 1 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octet</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 7 6 5 4 3 2 1</td>
</tr>
</tbody>
</table>

Figure 5.6-3: SIMP/L LAP/D Frame Specifications (MOD 128)

*The SI0 and SI1 commands and responses are not yet supported by the Chameleon.*
SIMP/L LAPD PARAMETER SET-UP MENU

Introduction

This section describes the parameters and valid values for the SIMP/L LAPD menu. This menu enables you to configure and save parameters for running SIMP/L LAPD simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu (option S), refer to Section 2.3.

Figure 5.6-4: SIMP/L LAPD Parameter Set-Up Menu

The SIMP/L LAPD parameters and listed below. A description of each parameter is on the next page.

A. Simulate
   DCE
   DTE

B. Data Encoding
   NRZ
   NRZI

C. Bit Rate
   Range: 50 - 64000 bits (DCE)
   RECEIVED (DTE)

D. Simulate
   Network
   Subscriber

E. Interframe fill
   7E hex
   FF hex

F. SIMP/L Type
   Extended
   Non-extended
If you are unfamiliar with these parameters, use the following guidelines for assistance in selecting the correct settings.

**Simulate:**
**DCE/DTE**

The DCE furnishes the clock which determines the data rate. It also determines on which connector pins the data and clocks are physically sent and received. For Simulation, one device is designated as the DCE; the other device is designated as a DTE.

If the Unit Under Test (UUT) is supplying the clock, it is the DCE. In this case, the Chameleon must be configured as a DTE. If the UUT is not supplying the clock, it is designated as a DTE. The Chameleon must then be configured as the DCE.

**Data Encoding:**
**NRZ/NRZI**

This parameter determines if the data is transmitted in NRZ (Non-Return to Zero) or NRZI (Non-Return to Zero Inverted) format. This is a function of the relative voltage on the data line, representing the 'MARK' and 'SPACE' levels.

Configure the Chameleon to use the same data encoding format as the (UUT).

**Simulate:**
**Network Subscriber**

This parameter determines if the Chameleon is acting as a subscriber (terminal) or as a network device. One device (either the UUT or the Chameleon) is designated the network device and the other device is designated the subscriber device.

This setting determines the value of the Command/Response (C/R) bit. The C/R bit is set to 1 for commands from and responses to the network side of the link. The C/R bit is set to 0 for commands from and responses to the subscriber side of the link.

A device can function as both a network and a subscriber device depending on its communications function with the other devices in the network. For example, a PBX is a subscriber device in the link between the PBX and the local switch. The same PBX is a network device in the link with a users terminal.

If the UUT is acting as a network device, configure the Chameleon as a subscriber device. If the UUT is acting as a subscriber device, configure the Chameleon as a network device. This parameter can be changed under program control using the SET command so that you can test the ability of the UUT to ignore commands from like devices.
Interframe Fill: 7E/FF

The Interframe fill determines what the Chameleon sends to the UUT when the data line is idle. The original CCITT recommendation specified an all 1's (FF) condition to indicate the line is idle and 7E if the line is busy.

Many terminal devices require an interframe fill value of FF since an interframe fill of 7E can be mistaken as the beginning and ending flags of an endless stream of zero-length frames. If the interframe fill is mistaken in this way, the terminal becomes so busy that it appears to be locked up.

Many switch devices assume that the physical link is disconnected or faulty unless there is a continuous stream of bits consisting of data or hex 7E flags on the line.

Note

This parameter is displayed for both Extendable and Non-Extended SIMP/L LAPD; however, it functions only in Extendable SIMP/L LAPD.

For both Extendable and Non-Extended SIMP/L LAPD, this parameter can be changed under program control using the \texttt{SET FILL=FF} or \texttt{SET FILL=7E} commands. Refer to page 5.6-21 for more information.

SIMP/L Type: Extended/Non-Extended

This option is displayed only if you have a Chameleon with extended memory. If you have a Chameleon with extended memory, the Extendable version of SIMP/L LAPD is automatically loaded. If you access the parameter set-up menu, this parameter appears in the menu. It gives you the option of using the Extendable SIMP/L LAPD in an extended or non-extended mode.

If you do not have extended memory, Non-Extended SIMP/L LAPD is automatically loaded. This menu option does not appear in the parameter set-up menu, because you can only use the Non-extended version of the software.

Note

SIMP/L LAPD and Multi-Link SIMP/L LAPD files reside in the same hard disk directory and use the same file extensions. This was done to simplify the task of modifying programs to work with either simulator. However, you will want to create separate parameter set-up files for your SIMP/L LAPD and Multi-Link LAPD programs.

If a Multi-Link parameter set-up file is used in SIMP/L LAPD, this parameter may be modified to Non-Extended. This may result in errors if that same parameter set-up file is then used with the Multi-Link LAPD simulator.
SIMP/L LAPD COMMAND INDEX

Introduction

This section lists the SIMP/L LAPD commands by function. For more information about a specific command, refer to the page number indicated.

Remember that you can also use the SIMP/L commands in Section 5.3 and the BASIC commands in Section 3.5.

TABLE 5.6-5: SIMP/L LAPD READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRSTAT</td>
<td>5.6-14</td>
<td>Variable</td>
<td>Returns the frame status byte.</td>
</tr>
<tr>
<td>LNKSTAT</td>
<td>5.6-15</td>
<td>Variable</td>
<td>Returns a value that indicates the status of the link.</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.6-26</td>
<td>Variable</td>
<td>Displays values of variables and status of the link.</td>
</tr>
</tbody>
</table>

TABLE 5.6-6: SIMP/L LAPD PARAMETER COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTEND</td>
<td>5.6-13</td>
<td>S or D</td>
<td>Selects Extended version of SIMP/L LAPD simulator if extended memory is present.</td>
</tr>
<tr>
<td>MOD</td>
<td>5.6-17</td>
<td>S or D</td>
<td>Sets either MOD8 or MOD128 sequencing.</td>
</tr>
<tr>
<td>SET</td>
<td>5.6-18</td>
<td>S or D</td>
<td>Assigns values to variables and timers.</td>
</tr>
<tr>
<td>SET {fnctn}</td>
<td>5.6-21</td>
<td>S or D</td>
<td>Sets individual control options in the control configuration byte.</td>
</tr>
<tr>
<td>SET CONFIG</td>
<td>5.6-22</td>
<td>S or D</td>
<td>Inserts a value into the control configuration byte.</td>
</tr>
<tr>
<td>SET RSAPI</td>
<td>5.6-24</td>
<td>S or D</td>
<td>Assigns values to user-defined SAPIs.</td>
</tr>
<tr>
<td>SET RTEI</td>
<td>5.6-25</td>
<td>S or D</td>
<td>Assigns values to user-defined TEIs.</td>
</tr>
<tr>
<td>UNEXTEND</td>
<td>5.6-32</td>
<td>S or D</td>
<td>Selects non-extended version of SIMP/L LAPD simulator if extended memory is present.</td>
</tr>
</tbody>
</table>
### TABLE 5.6-7: SIMP/L LAPD TRANSMISSION COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPRINT</td>
<td>5.6-27</td>
<td>S or D</td>
<td>Displays contents of trace buffer.</td>
</tr>
<tr>
<td>TRUI</td>
<td>5.6-29</td>
<td>S or D</td>
<td>Transmits an unnumbered I-frame.</td>
</tr>
<tr>
<td>TRXIDC</td>
<td>5.6-30</td>
<td>S or D</td>
<td>Transmits an XID command frame.</td>
</tr>
<tr>
<td>TRXIDR</td>
<td>5.6-31</td>
<td>S or D</td>
<td>Transmits an XID response frame.</td>
</tr>
</tbody>
</table>
*EXTEND

Description
Extensible SIMP/L LAPD only. The EXTEND command enables you to change from the non-extended to the extensible version of the SIMP/L LAPD simulation software.

Extensible SIMP/L LAPD requires extended RAM and has additional features that are not available in the non-extended version. See pages 5.6-1 through 5.6-4 for a general description of these features.

If you attempt to use the EXTEND command (or any other extensible SIMP/L LAPD command) and you do not have extended memory in your Chameleon, the command will be ignored.

Type
Statement or Direct

Syntax
EXTEND

Abbreviation
EXT

See Also
UNEXTEND

Example
EXTEND

TEKELEC
5.6-13
6/11/90
FRSTAT

Description
FRSTAT is a read-only variable that returns the frame status byte of the last received data packet. The frame status byte is added to the beginning of each received data packet and provides the following information:

- Frame type
- SAPI and TEI that were matched
- C/R bit value
- Poll/Final bit value

Using FRSTAT is easier than using the BREAK command when you need to extract only the frame status byte. If you are using SIMP/L LAPD in the UNEXTEND mode, you must use FRSTAT to access the frame status byte, since in this mode it is hidden from the program. In EXTEND mode, the frame status byte is displayed in using the TPRINT command.

The figure below shows the status byte interpretation.

```
7 6 5 4 3 2 1 0
```

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

High bit/low bit

- 00 = UI frame
- 01 = XID frame
- 10 = L-frame

- 00 = Matched RTE10
- 01 = Matched RTE11
- 10 = Matched RTE12
- 11 = Broadcast TEI

- 00 = Matched RSAPI0
- 01 = Matched RSAPI1
- 10 = Matched RSAPI2

- 0 = Command frame
- 1 = Response frame

- 0 = Poll/Final bit clear
- 1 = Poll/Final bit set

Type
Read-only variable

Syntax
(See example below.)

Abbreviation
FRS.

See Also
BREAK, TPRINT, EXTEND, UNEXTEND

Example
PRINT FRSTAT

TEKELEC 5.6-14 6/11/90
**LNKSTAT**

**Description**

LNKSTAT is a read-only variable that returns a value between 0 and 8 indicating the status of the link, as shown in the table below. Note that state 8 (Remote Station Not Responding) is in Extendable SIMP/L LAPD only.

<table>
<thead>
<tr>
<th>LNKSTAT</th>
<th>LINK STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link Disconnected Mode</td>
</tr>
<tr>
<td>1</td>
<td>Link Connection Requested</td>
</tr>
<tr>
<td>2</td>
<td>Frame Rejected</td>
</tr>
<tr>
<td>3</td>
<td>Disconnect Requested State</td>
</tr>
<tr>
<td>4</td>
<td>Information Transfer State</td>
</tr>
<tr>
<td>5</td>
<td>Local Station Busy</td>
</tr>
<tr>
<td>6</td>
<td>Remote Station Busy</td>
</tr>
<tr>
<td>7</td>
<td>Local and Remote Stations Busy</td>
</tr>
<tr>
<td>8*</td>
<td>Remote Station Not Responding</td>
</tr>
</tbody>
</table>

Table 5.6-8: SIMP/L LAPD Link Status Values

Since an XID frame can be used to monitor the physical link in single frame mode, state 8 occurs when N200 XID frames have not been answered when XID monitor mode is selected.

Internally, state 8 is treated the same as state 0, with recovery occurring when any acceptable frame is received from the remote station. If the Chameleon is in state 4 - 7 when the remote station stops responding, the following process occurs:

1. N200 attempts are made to exchange XID frames with the remote station.
2. State 1 is entered and N200 attempts are made to re-establish the link.
3. State 0 is entered and a final N200 XIDs are sent as a last attempt to re-establish the link.
4. State 8 is entered.

**Type**

Read-only variable
Syntax (See examples below.)

Abbreviation LN.

See Also STATUS

Example 1 This example displays the current LNKSTAT value.
PRINT LNKSTAT

Example 2 This example prints a message indicating the link status.
10 IF LNKSTAT=4 PRINT "INFORMATION TRANSFER STATE"
**MOD**

**Description**  The MOD command sets the modulus of the N(R) and N(S) fields to either MOD 8 or MOD 128. MOD 8 uses a range from 0 - 7. MOD 128 uses a range from 0 - 127.

**Type**  Statement or Direct

**Syntax**  
- MOD8
- MOD128

**Abbreviation**  None

**See Also**  STATUS

**Example**  This example set the modulo to 128.
- MOD128
SET

Description  In SIMP/L LAPD, the SET command can set the following variables:

- N200
- N201
- Network or Subscriber
- SAPI
- TEI
- T200
- T203
- Window

Each of these variables is described below.

N200  The N200 variable defines the maximum number of frame retransmissions that can occur after successive lapses of the T201 timer before declaring the link unattainable, and sending polled disconnects. N200 must not be set to zero.

N201  The N201 variable defines the maximum number of bytes in a frame. The N201 variable has a valid range from 2 - 512 bytes, inclusive. When the system receives or transmits a frame, it checks the value of the N201 variable.

If the system receives a frame longer than N201, it automatically sends a frame reject (FRMR).

NETWORK/SUBSCRIBER  You can simulate either a network or subscriber device.

When the Chameleon emulates a LAPD network, it sends commands with the C/R bit set to one, and responds with the C/R bit set to zero. It sends the selected SAPI and TEI with the C/R bit automatically set in accordance with CCITT Q. 921.

When the Chameleon emulates a LAPD subscriber, it sends commands with the C/R bit set to zero, and responds with the C/R bit set to one. It sends the selected SAPI and TEI with the C/R bit automatically set in accordance with CCITT Q. 921.

SAPI  The SAPI (Service Access Point Identifier) indicates the layer two service type requested or supported. Normal values are:

- 0  Call Control procedures
- 16  Packet communication procedures
- 63  Management procedures
TEI

The TEI (Terminal Endpoint Identifier) is a value assigned to and may be associated with a single terminal and a given point-to-point data link connection. At any time, a given terminal endpoint (TE) may contain one or more TEIs.

This value may be assigned by the carrier at the time of equipment installation, or may be automatically assigned on a call-by-call basis. The broadcast value is associated with all user-side data link entities with the same SAPI, regardless of other assigned value(s).

Normal values are:

- 0 - 63 Non-Automatically assigned values
- 64 - 126 Automatically assigned values
- 127 Broadcast value

T200

The T200 variable defines the maximum timeout period that can elapse between sending a frame and receiving an acknowledgment. The timer is started by the station when it sends any command frame requiring a response.

If T200 expires before the station receives the expected response, it retransmits the message and restarts the timer. The transmission of an I-frame also restarts the T200 timer.

When T200 expires, the frame is retransmitted N200 times at T200 intervals. Following an N200 number of retransmissions, the station takes the appropriate recovery action.

If T200 expires and outstanding frames remain unacknowledged, the station re-arms T200 and sends an appropriate Supervisory Command Frame with the poll bit set.

T200 is expressed in tenths of seconds. The T200 timer can be disabled by setting it to zero. Disabling the T200 timer enables you to test a device without receiving a retransmitted frame while performing a manual operation or test measurement.

T203

The T203 variable defines the maximum amount of time allowed between the transmission of frames. If this timer expires, the Chameleon tests the link conditions by transmitting an RR, RNR, REJ or XID command, depending on the current state and configuration.

The T203 timer can be disabled by setting it to zero. Disabling the T203 timer enables you to test a device without receiving a link status check while performing a manual operation or test measurement.
Window

The \textit{WINDOW} variable defines the maximum number of sequentially numbered I-frames that the DCE or DTE can have outstanding (unacknowledged) at any given time. \textit{WINDOW} has a range of 1 - 7.

Type

Statement or Direct

Syntax

\begin{verbatim}
SET N200 = x
SET N201 = x
SET SAPI = x
SET TEI = x
SET T200 = x
SET T203 = x
SET Window = x
\end{verbatim}

where: \textit{exp} is a number, variable or expression that is valid for that variable.

Abbreviation

None

See Also

\textit{STATUS}

Example 1

This example sets several variables and then displays their values using the \textit{STATUS} command.

\begin{verbatim}
10 SET N201 = 20
20 SET SUBSCRIBER
30 SET NETWORK
40 SET WINDOW = 3
50 STATUS
\end{verbatim}

Example 2

This example sets the value of N201 (maximum frame size) and then attempts to build a frame using a string that exceeds the maximum. This results in an error message.

\begin{verbatim}
10 $A = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 SET N201 = 10
30 BUILD $A
\end{verbatim}

\textbf{Result} \hspace{1cm} \texttt{Overflow!}
**SET {fnctn}**

**Description**  Extendable SIMP/L LAPD only. The SET {fnctn} command sets individual control options in the control configuration byte.

**Type**  Statement or Direct

**Syntax**  

\[ \text{SET } \text{fnctn} \]

*where: fnctn is a MNEMONIC in the table below that selects a specific control option. Note that spaces are not allowed within a mnemonic.*

**Abbreviation**  None

**See Also**  SET CONFIG

**Example**  

To Poll only XID frames with I-fields, enter:

\[ \text{SET POLIXID} \]

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>CONTROL FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTRICT</td>
<td>Restrict received responses to the transmit SAPI and TEI values.</td>
</tr>
<tr>
<td>UNRESTRICT</td>
<td>Accept responses matching user-defined SAPIs and TEIs and broadcast TEI.</td>
</tr>
<tr>
<td>SBMCOL</td>
<td>Generate SABM(E) collisions.</td>
</tr>
<tr>
<td>NOSBMCOL</td>
<td>Stop generating SABM(E) collisions.</td>
</tr>
<tr>
<td>XIDEXCH</td>
<td>Transmit XID command on T203 timeout.</td>
</tr>
<tr>
<td>NOXIDEXCH</td>
<td>Stop transmitting XID's on T203 timeout.</td>
</tr>
<tr>
<td>POLLSTCH</td>
<td>Set poll bit on status changing frames SABM(E) and DISC.</td>
</tr>
<tr>
<td>NORMSTCH</td>
<td>Set poll bit normal on status changing frames SABM(E) and DISC.</td>
</tr>
<tr>
<td>POLALXID</td>
<td>All XID frames polled.</td>
</tr>
<tr>
<td>ALXI DNPL</td>
<td>All XID frames not polled.</td>
</tr>
<tr>
<td>POLIXID</td>
<td>Poll only XID frames with I-fields.</td>
</tr>
<tr>
<td>POLIXID</td>
<td>Poll only XID frames without I-fields.</td>
</tr>
<tr>
<td>FILL = 7E</td>
<td>Set interframe fill to value 7E.</td>
</tr>
<tr>
<td>FILL = FF</td>
<td>Set interframe fill to value FF.</td>
</tr>
</tbody>
</table>

Figure 5.6-9: User Control Mnemonics
**SET CONFIG**

**Description**

Extendable SIMP/L LAPD only. SET CONFIG sets all control option values using a single command. It inserts a hex value into the bit-mapped control configuration byte shown in the figure below.

The default setting is as follows:

A. Poll bit normal on state change frames.
B. Poll bit normal on XID frames.
C. SABM(E) collisions not generated.
D. Exchange polled RR (RNR) frames as physical link test.
E. Accept response frames matching any receive SAPI/TEI combination.
F. Set the interframe fill as selected in the setup menu.

![Control Configuration Byte](image)

Figure 5.6-10: Control Configuration Byte
<table>
<thead>
<tr>
<th>Type</th>
<th>Statement or Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>SET CONFIG = xx</td>
</tr>
<tr>
<td></td>
<td>where: xx is a hex value.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>None</td>
</tr>
<tr>
<td>See Also</td>
<td>SET {fnctn}</td>
</tr>
<tr>
<td>Example</td>
<td>This example sets the value of the control configuration byte to 7A hex.</td>
</tr>
<tr>
<td></td>
<td>SET CONFIG = 7A</td>
</tr>
<tr>
<td></td>
<td>The hex value 7A is represented in binary as:</td>
</tr>
<tr>
<td></td>
<td>0111,1010</td>
</tr>
<tr>
<td></td>
<td>which sets the status configuration byte as follows:</td>
</tr>
<tr>
<td>Bit 7</td>
<td>0                   Stop transmitting XID frames on T203 timeout</td>
</tr>
<tr>
<td>Bit 6,5</td>
<td>11                  Poll all XID frames</td>
</tr>
<tr>
<td>Bit 4</td>
<td>1                   Generate SABM(E) collisions</td>
</tr>
<tr>
<td>Bit 3</td>
<td>1                   Poll set</td>
</tr>
<tr>
<td>Bit 2</td>
<td>0                   Unrestricted address</td>
</tr>
<tr>
<td>Bit 1</td>
<td>1                   Interframe fill = FF</td>
</tr>
<tr>
<td>Bit 0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
*SET RSAPI

Description
Extendable SIMP/L LAPD only. The SET RSAPI command sets the values of a user-defined receive SAPI. Three user-defined receive SAPIs can be active at any one time.

Type
Statement or direct

Syntax
SET RSAPI\textsubscript{n} = x

where: \( n \) is the SAPI number in the range 0 - 2 and \( x \) is the SAPI value in the range 0 - 255. User-defined SAPIs have the following default values:

- RSAPI0 = 0 (Call control procedures)
- RSAPI1 = 16 (Packet communications procedures)
- RSAPI2 = 63 (Management procedures)

To disable a user-defined SAPI, use the SET RSAPI command to assign an invalid value to the SAPI, or assign a SAPI value that is already in use. Note that the SET SAPI command sets the SAPI for transmitted commands.

See Also
SET RTEI, SET SAPI

Example
This example sets user-defined SAPI 1 to a value of zero.

SET RSAPI1 = 0
*SET RTEI

Description
Extendable SIMP/L LAPD only. The SET RTEI command sets the values of a user-defined receive TEI. Three user-defined receive TEIs can be active at any one time.

Type
Statement or Direct

Syntax
SET RTEIn = x

where: n is the TEI number in the range 0 - 2 and x is the TEI value in the range 0 - 255 decimal. The default value for all user-defined TEIs is 127 (broadcast). Note that the SET TEI command sets the TEI for transmitted commands.

See Also
SET RSAPI, SET TEI

Example
This example sets the user-defined receive TEI number 0 to a value of 100.

SET RTEI0 = 100
STATUS

Description The STATUS command displays the following information:

- Link status
- T200
- T203
- Program Mode (Extended/Non-extended):
- Current SAPI
- Current TEI
- N201
- N200
- WINDOW
- Modulo
- Subscriber or Network
- Active SAPs: values of RSAPI0, RSAPI1, and RSAPI2
  (Extendable SIMP/L LAPD only)
- Active TEIs: values of RTEI0, RTEI1, RTEI2, and
  broadcast TEI (Extendable SIMP/L LAPD) or last received
  TEI matched/broadcast (SIMP/L LAPD)

Type Statement or Direct

Syntax STATUS

Abbreviation STA.

See Also SET, SET RSAPI, SET RTEI, MOD

Example In Extendable SIMP/L LAPD, the STATUS command results in
a display with the following format:

!STATUS

Link-Disconnected
Timer T200 Value - 10 Timer T203 Value - 20
Program Mode-Extended
Current SAPI Value - 0
Current TEI Value - 0
Maximum frame size (N201) - 260
Number of Re-transmissions (N200) - 3
Window - 3 MOD128
Simulating a network.
Accepting frames with these SAPI’s
0, 16, 63
And these TEI’s
115, 27, 102, 127
TPRINT

Description
TPRINT displays the contents of the trace buffer, which contains the frames transmitted and received. Refer to Section 3.2 for a general description of the trace buffer.

The contents of the trace buffer depends on whether you are using Extendable SIMP/L LAPD or non-extended SIMP/L LAPD. The Extendable SIMP/L LAPD trace buffer includes the frame status byte which indicates the type of frame received.

The format of the trace display for both SIMP/L LAPD versions is illustrated in the examples below.

Type
Statement or Direct

Syntax
TPRINT

Abbreviation
TP.

See Also
TSAVE, TLOAD, FLUSH, TFREE

Example

```
R 00:00:00:00:00:00  FRSTAT 02  LEN  1234  08  01  A3  05  04  88  90  80  80
T 00:00:00:00:00:00  LEN  4321  08  01  A3  0D
```

Time Stamp:
Hours: Minutes: Seconds: .01 Seconds: .0001 Seconds

Extensible SIMP/L LAPD Trace Display
<table>
<thead>
<tr>
<th>T</th>
<th>Transmitted frame</th>
<th></th>
<th>R</th>
<th>Received frame</th>
<th></th>
<th>Frame Length</th>
<th>4 digit, right justified</th>
<th>Contents of received data packet - in hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>R 00:00:00:00:00 LEN 1234 08 01 A3 05 04 88 90 80 80</td>
<td></td>
<td></td>
<td>T 00:00:00:00:00 LEN 4321 08 01 A3 0D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time Stamp:
Hours: Minutes: Seconds:01 Seconds: .0001 Seconds

Non-Extended SIMPL LAPD Trace Display
<table>
<thead>
<tr>
<th>Description</th>
<th>TRUI transmits an unnumbered I-frame (UI frame) built using the BUILD command. Refer to Tables 5.6-1 and 5.6-2 for descriptions of UI frame formats.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>TRUI</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>TRU.</td>
</tr>
<tr>
<td>See Also</td>
<td>BUILD</td>
</tr>
<tr>
<td>Example</td>
<td>This example builds and transmits a UI frame.</td>
</tr>
<tr>
<td></td>
<td>10 $A = &quot;HELLO&quot;</td>
</tr>
<tr>
<td></td>
<td>20 BUILD $A</td>
</tr>
<tr>
<td></td>
<td>30 TRUI</td>
</tr>
</tbody>
</table>
*TRXIDC

Description: Extendable SIMP/L LAPD only. TRXIDC transmits an XID command frame built using the BUILD command. Refer to Tables 5.6-1 and 5.6-2 for descriptions of UI frame formats.

Type: Statement or Direct

Syntax: TRXIDC

Abbreviation: None

See Also: TRXIDR, BUILD

Example: This example builds and transmits a frame using TRXIDC.

10 $A = "HELLO"
20 BUILD $A
30 TRXIDC
**TRXIDR**

| **Description** | Extendable SIMP/L LAPD only. TRXIDR transmits an XID response frame built using the BUILD command. Refer to Tables 5.6-1 and 5.6-2 for descriptions of UI frame formats. This TRXIDR command transmits an XID response frame with a copy of the last received SAPI and TEI. In this way, the process of preparing the SAPI-C/R combination takes less time and the remote unit receives the response it is expecting. In light of this, it is highly recommended that SIMP/L LAPD application programs be designed to monitor received frames and respond immediately when an XID command is received. If an XID response is sent at a different time, the remote unit may treat it as a command, if the last received frame was a response. If an XID command is received without an I-field, it is assumed to be a link monitor frame. In this case, an XID response with matching SAPI and TEI is automatically transmitted without an I-field. This relieves the programmer of having to generate responses in applications which use XID frames to test the physical link. |
| **Type** | Statement or Direct |
| **Syntax** | TRXIDR |
| **Abbreviation** | None |
| **See Also** | TRXIDC, BUILD |
| **Example** | This example builds and transmits a frame using TRXIDR. |

```
10 $A = "HELLO"
20 BUILD $A
30 TRXIDR
```
**UNEXTEND**

**Description**
Extendable SIMP/L LAPD only. The UNEXTEND command enables you to change from the Extended to the non-extended mode when Extendable SIMP/L LAPD is loaded on your Chameleon.

Extendable SIMP/L LAPD requires extended RAM and has additional features that are not available in the non-extended version. See pages 5.6-1 through 5.6-4 for a general description of these features.

If you attempt to use the UNEXTEND command (or any other Extendable SIMP/L LAPD command) and you do not have extended memory in your Chameleon, the command will be ignored.

**Type**
Statement or Direct

**Syntax**
UNEXTEND

**Abbreviation**
UNE.

**See Also**
EXTEND

**Example**
UNEXTEND
SIMP/L LAPD MNEMONICS

Introduction
This section describes the default mnemonic table for SIMP/L LAPD. For a general description of the use of mnemonic tables refer to Section 3.2.

Q.931 Message
The general format of a Q.931 message is illustrated in the figure below.

8 7 6 5 4 3 2 1 OCTET

<table>
<thead>
<tr>
<th>Protocol Discriminator</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 Call Ref Length</td>
<td>2</td>
</tr>
<tr>
<td>Call Reference Value</td>
<td>3</td>
</tr>
<tr>
<td>0 Message Type</td>
<td>4 (If call Red Length = 1)</td>
</tr>
<tr>
<td>0/1 Mandatory and optional information elements</td>
<td>etc.</td>
</tr>
</tbody>
</table>

Figure 5.6-11: General Q.931 Message Format
Entries

The SIMP/L LAPD mnemonic table (shown below) has two columns:

- Mnemonic name (1 - 6 characters)
- Field width in bits

The figure below also provides mnemonic definitions and locations in the Q.931 message. Using the BASIC mnemonic commands described in Section 3.5, you can manipulate the default table or define and save your own mnemonic tables.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>FIELD WIDTH (BITS)</th>
<th>DEFINITION/ Q.931 MESSAGE OCTET</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESTYP</td>
<td>7</td>
<td>Message type/fourth octet</td>
</tr>
<tr>
<td>SHF1D</td>
<td>3</td>
<td>Shift 10/fourth octet (Shift info. element)</td>
</tr>
<tr>
<td>LOKBIT</td>
<td>1</td>
<td>Shift lock bit/fourth octet (Shift info. element)</td>
</tr>
<tr>
<td>CODSET</td>
<td>3</td>
<td>Code set/fourth octet (Shift info. element)</td>
</tr>
<tr>
<td>CRLEN</td>
<td>4</td>
<td>Call reference length/second octet</td>
</tr>
<tr>
<td>CREF7</td>
<td>7</td>
<td>Call reference/third octet</td>
</tr>
<tr>
<td>CREF8</td>
<td>8</td>
<td>Call reference/third octet</td>
</tr>
<tr>
<td>NOEXT</td>
<td>1</td>
<td>No extended bit/fourth octet filler</td>
</tr>
<tr>
<td>PDIS</td>
<td>8</td>
<td>Protocol discriminator/first octet</td>
</tr>
<tr>
<td>FIL4</td>
<td>4</td>
<td>Four bit filler/second octet</td>
</tr>
<tr>
<td>PAD1</td>
<td>1</td>
<td>One bit filler/fourth octet filler</td>
</tr>
<tr>
<td>PAD2</td>
<td>2</td>
<td>Two bit filler</td>
</tr>
<tr>
<td>EXT</td>
<td>1</td>
<td>Extend bit</td>
</tr>
<tr>
<td>RI</td>
<td>16</td>
<td>Reference number/TEI field</td>
</tr>
<tr>
<td>AI</td>
<td>7</td>
<td>Action indicator/TEI field</td>
</tr>
</tbody>
</table>

Table 5.6-12: SIMP/L LAPD Mnemonics
FRAME LEVEL CONFIGURATION

This section provides an introduction to SIMP/L LAPD frame level configuration and is organized into the following sections, for easier reference:

- Physical link monitoring
- Information Transfer state
- SAPI and TEl control
- Packet Status Byte

Physical Link Monitoring

If no frames are exchanged within the time established by the T203 timer, a supervisory frame is exchanged to determine if the physical link is still established. The type of supervisory frame that is exchanged is determined by the conditions described below.

Polled RR command and response frames are exchanged when in Information Transfer mode. This occurs automatically unless another mode is selected.

You can inhibit link monitoring frames from being generated by the Chameleon by disabling the T203. This is done by setting the T203 timer to 0. The Chameleon will still respond to link monitoring frames sent by the Unit Under Test (UUT).

XID command and response frames can be exchanged in all link states except state value 8 (remote unit not responding). The command SET XIDEXCH initiates transmission of XID commands without I-Fields each T203 timeout.

If the UUT fails to respond to the XIDEXCH command, it is repeated N200 times. If the UUT has still not responded, and the link was in Information Transfer mode, the Chameleon attempts to re-establish the link by sending SABM(E)s.

If the UUT still does not respond, the Chameleon goes to disconnected state, tries again to establish contact with XID frames, then enters the remote unit not responding state (8). Frames are handled the same in state 8 as in state value 0 (disconnected) and any received frame sets the state to 0.
If the UUT requires an XID command and response to be exchanged in the Information Transfer state, you can write a program similar to the following:

10 SLON

20 IF LNKSTAT = 1 GOTO 20

30 IF LNKSTAT#4 GOTO 100

40 SET XIDEXCH

...more program steps...

100 PRINT "LINK NOT ESTABLISHED"

The following program illustrates a technique that can be used if the UUT requires XID exchanges only in the TEI assigned state, and RR exchanges to monitor the link in the Information Transfer state.

These lines follow the program lines that request or assign a TEI. See page 4.8-25 for a sample program that does this.

100 REM TEI IS ASSIGNED

110 SET XIDEXCH

...more program steps...

300 SLON

310 IF LNKSTAT = 1 GOTO 310

If in Link Connection Requested state, waits until finished.
320 IF LNKSTAT#4 GOTO 1000  
Determines if link is in Information Transfer state.

330 SET NOXIDEXCH  
Switches to RR exchange.

Information Transfer State

The following parameters are described in this section:

- Poll bit control
- Poll/Final bit control in XID frames
- SABM(E) Collisions

Poll Bit Control

The poll bit is set to 0 on the first transmission of a SABM(E) and retransmissions polled. This is the normal mode, which occurs when the SLON (Set Link ON) command is used.

The poll bit is set to 1 on all transmissions of SABM(E) and polling DISC frames when the SET POLSTCH command is used. The command SET NORMSTCH cancels this feature.

You can configure the Poll bit in XID command frames for all possible combinations using the following commands:

```
SET POLALXID     Polls all XID frames
SET ALXIDNPL      Does not poll any XID frames
SET POLIXID       Polls XID frames with I-fields
SET POLNIXID      Polls XID frames without I-Fields
```

The SET SBMCOL commands enables you to generate SABM(E) collisions. Some devices will not enter Information Transfer state after sending a SABM(E) unless a SABM(E) frame and a UA frame are received. That is, each side must receive a SABM(E) and a UA in answer to its SABM(E) to set up Information Transfer.

The SET NOSBMCOL command cancels this feature.
You can program the Chameleon to accept frames matching as many as three SAPIs and three TEIs and the broadcast TEI.

If fewer than three SAPIs or TEIs are needed, the unused SAPIs and TEIs are disabled, using one of two methods:

1. Set more than one SAPI or TEI to the same value
2. Set unused SAPIs and TEIs to invalid values, for example:

   SET RSAPI2 = 200

To accept command frames matching any combination of user defined SAPI and TEI, but restrict accepted response frames to the transmitted SAPI and TEI values, use the command SET RESTRICT. This feature is canceled by the SET UNRESTRICT command.

The SAPI and TEI values transmitted in command frames are set independently from the accepted (receive) values. Use the following commands to set SAPI and TEI values for command frames:

   SET SAPI = nn  (nn = 0 - 63)
   SET TEI = nn   (nn = 0 - 127)

If a command is received with a TEI and SAPI matching the acceptable receive values, and the command requires an automatic response (for example, a SABME is received), the response automatically sets the SAPI and TEI to match the received command.

No automatic response is generated for XID commands with I-fields because the format and content of such frames varies depending on the implementation. It is recommended that your programs closely monitor frame reception and respond appropriately when an XID frame with an I-field is received.

Received data packets are passed to your program with a status byte prefix which contains the following information:

- Type of frame received
- SAPI and TEI used,
- State of the poll/final bit
In many applications, XID commands and responses are used for handshaking to set up layer 2 parameters. In these cases, it is important to know if an XID was a command or response.

The following program uses the frame status byte. It looks for an XID command frame matching RSAPI0 and RTEI1 and performs a special action when a frame that matches is received.

10 DEFINE "FSTAT" = 8 Defines FSTAT as a mnemonic for an 8 bit value.

20 REC Receives a packet.

30 IF LENGTH = 0 GOTO 10 If no data is received, loops to try again.

40 BREAK FSTAT,SA Breaks the received packet into two parts. The first 8 bits (the status byte) goes into FSTAT; the packet contents go into $A.

50 A = FSTAT AND &7F Logically ANDs the status byte with hex 7F to remove the Poll/Final indicator, and stores the result in numeric variable A.

60 IF A & 05 GOTO 1000 Tests the status byte for a binary value 0000 0101 (hex 05) to determine if the following condition exists, and goes to line 1000 if true:

BIT 7 = 0 Forced true by line 50
BIT 6 = 0 = Command frame
BIT 5,4 = 00 RSAPI0 matched
BIT 3,2 = 01 RTEI1 matched
BIT 1,0 = 01, Frame = XID

70 GOTO 20 Loops back to receive another frame if this was not the desired one.
Table 5.6-9 summarizes the SIMP/L LAPD parameters.

<table>
<thead>
<tr>
<th>IF THE UNIT UNDER TEST IS:</th>
<th>SET THE Chameleon TO:</th>
<th>CONFIGURE FROM:</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTE</td>
<td>DCE</td>
<td>MENU</td>
<td></td>
</tr>
<tr>
<td>DCE</td>
<td>DTE</td>
<td>MENU</td>
<td></td>
</tr>
<tr>
<td>NRZ</td>
<td>NRZ</td>
<td>MENU</td>
<td></td>
</tr>
<tr>
<td>NRZI</td>
<td>NRZI</td>
<td>MENU</td>
<td></td>
</tr>
<tr>
<td>SUBSCRIBER</td>
<td>NETWORK</td>
<td>MENU/PROGRAM</td>
<td>SET NETWORK</td>
</tr>
<tr>
<td>NETWORK</td>
<td>SUBSCRIBER</td>
<td>MENU/PROGRAM</td>
<td>SET SUBSCRIBER</td>
</tr>
<tr>
<td>SABM(E) COLLISION REQUIRED</td>
<td>SABM(E) COLLISION MODE</td>
<td>PROGRAM</td>
<td>SET SBMCOL</td>
</tr>
<tr>
<td>SABM(E) COLLISION CAUSES TROUBLE</td>
<td>NO SABM(E) COLLISIONS</td>
<td>PROGRAM</td>
<td>SET NOSBMCOL</td>
</tr>
<tr>
<td>XID EXCHANGE ON T203 TIMEOUT</td>
<td>XID EXCHANGE</td>
<td>PROGRAM</td>
<td>SET XIDEXCH</td>
</tr>
<tr>
<td>RR EXCHANGE ON T203 TIMEOUT</td>
<td>NORMAL LINK</td>
<td>PROGRAM</td>
<td>SET NOXIDEXCH</td>
</tr>
<tr>
<td>ALL STATUS CHANGING FRAMES POLLED</td>
<td>POLL STATUS CHANGING FRAMES</td>
<td>PROGRAM</td>
<td>SET POLLSTCH</td>
</tr>
<tr>
<td>ALL STATUS CHANGING FRAME NOT POLLED</td>
<td>NORMAL STATUS CHANGING FRAMES</td>
<td>PROGRAM</td>
<td>SET NORMSTCH</td>
</tr>
<tr>
<td>XID FRAMES NOT POLLED</td>
<td>XID FRAMES NORMAL POLL</td>
<td>PROGRAM</td>
<td>SET ALXIDNPL</td>
</tr>
<tr>
<td>XID FRAMES POLLED</td>
<td>XID FRAMES ALWAYS POLLED</td>
<td>PROGRAM</td>
<td>SET POLALXID</td>
</tr>
<tr>
<td>XID FRAMES POLLED WITH I-FIELD</td>
<td>XID FRAMES POLLED IF INFO BEARING</td>
<td>PROGRAM</td>
<td>SET POLIXID</td>
</tr>
<tr>
<td>XID FRAMES POLLED WITH NO I-FIELD</td>
<td>XID FRAMES POLLED IF NOT INFO BEARING</td>
<td>PROGRAM</td>
<td>SET POLNIXID</td>
</tr>
<tr>
<td>MUST NOT RECEIVE LINK STATUS TEST EXCHANGE</td>
<td>DISABLE T203</td>
<td>PROGRAM</td>
<td>SET T203 = 0</td>
</tr>
<tr>
<td>MUST NOT RECEIVE RETRANSMITTED FRAMES</td>
<td>DISABLE T200</td>
<td>PROGRAM</td>
<td>SET T200 = 0</td>
</tr>
</tbody>
</table>
SIMP/L LAPD SAMPLE PROGRAMS

Introduction
The following examples of SIMP/L LAPD programs demonstrate major concepts to consider when developing applications. They are not valid tests for a particular application.

SLAPD1
The program demonstrates transmitting, receiving, and analyzing a Q.931 I-field with SIMP/L LAPD. It builds and transmits a message, asking the network for a TEI assignment. If the message is received, the TEI is assigned, and a congratulatory message is displayed. If the wrong message is received, an error message is displayed and the program stops.

All messages used for TEI assignment procedures are carried in the information field of UI command frames with a SAPI value set to 63 and TEI value set to 127.

Specifically, the program example does the following:

- Defines the lengths of the mnemonics
- Assigns values
- Assembles values for transmission
- Transmits values
- Checks the message type
5 CLEAR

10 DEFINE "MEI" = 8

20 DEFINE "RI" = 16

30 DEFINE "MESTYP" = 8

40 DEFINE "AI" = 8

50 SET TEI = 127

60 SET SAPI = 63

70 A = RND&FF

80 MEI = 15

90 RI = A

100 MESTYP = 0

110 AI = &FF

120 BUILD MEI, RI, MESTYP, AI

130 TRUI

140 REC

150 IF LENGTH = 0 GOTO 140

160 BREAK MEI, RI, MESTYP, AI

170 IF MESTYP = 2 GOTO 200

180 PRINT "WRONG PACKET RECEIVED...ERROR CONDITION"

190 STOP

200 PRINT "TEI ASSIGNED...CONGRATULATIONS"

210 STOP

Clears the trace buffer.

Sets Management Entity Identifier to 8 bits.

Sets Reference Number to 16 bits.

Sets Message Type to 8 bits.

Sets Action Indicator to 8 bits.

Sets Terminal End Point Identifier to 127.

Sets Service Access Point Identifier to 63.

Assigns a random number to A.

Sets Management Identifier to 15.

Sets Reference number to random number A.

Requests a TEI from the network.

Sets Action Indicator to 127 and message end bit, indicating that any TEI is acceptable.

Builds the frame.

Transmits the frame.

Receives a block.

If a block is received, analyzes it. If not, tries to receive again.

Breaks the received frame into mnemonics and bit widths.

If MESTYP = 2 goto 200

PRINT "WRONG PACKET RECEIVED...ERROR CONDITION"

STOP

PRINT "TEI ASSIGNED...CONGRATULATIONS"

STOP
SLAPD2

This program builds information frames for transmission. It checks the link status continually, transmitting only when in the proper state. When not in the proper state, it attempts to correct the problem and sets the link back to information transfer.

After going through all the transmission loops, it sets the link off, and prints an appropriate message.

Assign values to the address, enabling the link to be set to ON. Refer to Sample program 1 for a TEI request procedure.

Prompts the operator for modulo and converts response to uppercase.

Builds a frame using two string variables.

Changes the window value each time the transmission test is run.

Set link on for information transfer.

Creates a loop for the transmission of I-Frames twenty times.

If I-Transfer state, transmits and increments the loop.

If it is not an I-Transfer state, prints the status of the link.

Receives from the line, clearing the buffer of the local station, if necessary.

Checks the status of the link. If local station remote station, or local and remote station busy, enters the receive loop until the situation is corrected.
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>GOTO 230</td>
<td>Stations not busy--does not enter receive loop.</td>
</tr>
<tr>
<td>200</td>
<td>IF LENGTH=0 GOTO 170</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
<tr>
<td>210</td>
<td>TPRINT</td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>GOTO 170</td>
<td>Check frame length. If empty, receives again. If not empty, it prints before receiving another frame.</td>
</tr>
<tr>
<td>230</td>
<td>IF LMKSTAT=0 SLOF</td>
<td>If link status is disconnected, it attempts to set the link on.</td>
</tr>
<tr>
<td>240</td>
<td>GOTO 150</td>
<td>Keeps the program out of the transmit section, unless the link is an in l-Transfer state.</td>
</tr>
<tr>
<td>250</td>
<td>TRAN</td>
<td>Transmits the built frame.</td>
</tr>
<tr>
<td>260</td>
<td>NEXT A</td>
<td>Increments the transmit loop.</td>
</tr>
<tr>
<td>270</td>
<td>NEXT B</td>
<td>Increments the window loop.</td>
</tr>
<tr>
<td>280</td>
<td>SLOF</td>
<td>Sets the link off.</td>
</tr>
<tr>
<td>290</td>
<td>IF LMKSTAT #0 GOTO 290</td>
<td>Waits until the link is disconnected to continue.</td>
</tr>
<tr>
<td>300</td>
<td>?&quot;Test complete-link disconnected&quot;</td>
<td>Displays message that the link has been disconnected.</td>
</tr>
<tr>
<td>310</td>
<td>STATUS</td>
<td>Displays status of the link proving that it has been disconnected.</td>
</tr>
<tr>
<td>320</td>
<td>STOP</td>
<td>Ends program.</td>
</tr>
</tbody>
</table>
This is an example of Q.931 simulation of sending messages. It uses the DEFINE command to set up named variables, or mnemonics, to simplify setting up the message overhead and interpreting received messages.

10 DEFINE "PRODIS" = 8

Defines an 8-bit mnemonic to store the Protocol Discriminator.

20 DEFINE "CRFLEN" = 8

Defines an 8-bit mnemonic to store the Call Reference Length.

30 DEFINE "CRFIBY" = 8

Defines a 1-byte mnemonic for a Call Reference value.

40 DEFINE "CRF2BY" = 16

Defines a 2-byte mnemonic for a longer Call Reference value.

50 DEFINE "MESTYP" = 8

Defines an 8-bit mnemonic to store the Message Type.

60 PRODIS = 8

Assigns a value of 8 to the Protocol Discriminator.

70 CRFLEN = 1

Assigns a value of 1 to the Call Reference Length.

80 CRFIBY = V

Assigns the value of the numeric variable V to the Call Reference value.

90 MESTYP = N

Assigns the value of the numeric variable N to the Message Type.

100 $A = CHR$(&56)

Defines $A as an information element, indicating a locking shift to codeset 6. The predefined string variables $A through $Z are useful for setting up information elements.

200 BUILD PRODIS, CRFLEN, CRFIBY, MESTYP, $A, $B, $C

All parts are linked in the transmit buffer using the BUILD command.

210 TRAN

Transmits an I-Frame.

220 TRUI

Transmits a UI frame.

230 TRXIDC

Transmits an XID command frame.

240 TRXIDR

Transmits an XID response frame.
This is an example of Q.931 simulation which receives a message. When a message is received, the packet is preceded by a status byte that indicates the frame type. Once the frame type is identified, you can use the BREAK command to disassemble a message into its component parts.

In the example below, the status byte is stripped and interpreted, and the message is broken into appropriate components. It assumes that the mnemonics from sample program SLAPD3 have been defined in the mnemonic table.

10 DEFINE "FSTAT" = 8 Defines FSTAT as a mnemonic for an 8 bit value.

20 REC Receives a packet.

30 BREAK FSTAT, $A Breaks the received packet into two parts. The first 8 bits (the status byte) goes into FSTAT; the packet contents goes into $A.

Assuming that you received a Q.931 message, you can then use the BREAK with appropriate variables (see SLAPD3 sample program).

40 BREAK FSTAT, PRODIS, CRFLEN, CRF1BY, MESTYP, $Z

Separates the protocol discriminator, call reference parameters and message type so that they may be tested and acted upon. The actual message contents are stored in the string variable $Z.

Use pointers and the string functions to examine the contents of $Z. This example tests whether all the required information elements were included in the message.

90 T=0 Initializes variable T to count the total number of information elements.

100 P=1 Initializes variable P as a pointer to keep track of our place in the message.

110 C=0 Initializes variable C to count the number of correct information elements.

120 $X=MIDS($Z, P, 1) Extracts the information element ID value into $X using the MIDS function.

130 GOSUB 1000 Calls a subroutine to determine if the information element is one of those required by the message type.
140  P=P+1  
Moves the pointer to the information element length.

150  SX=MIDS(SZ,P,1)  
Extracts the information element length into SX using the MID function.

160  L=VAL(SX)  
Converts the length value to numeric variable L.

170  P=P+L+1  
Moves the pointer P to the next information element.

180  T=T+1  
Increments the total number of information elements.

190  IF  P< LEN(SZ) GOTO 120  
If pointer has not reached end of message, loops back.

200  ? "THERE WERE",C,"CORRECT INFO ELEMENTS, AND"

210  ? T-C,"INCORRECT INFO ELEMENTS."
Print a message displaying the test result.

220 STOP  
Stops the program. Lines 1000 to 1080 form the loop which tests whether the information element ID's are on the list of those required.

1000 IF SX=CHR$(&02) C=C+1  
1010 IF SX=CHR$(&08) C=C+1  
1020 IF SX=CHR$(&12) C=C+1  
1030 IF SX=CHR$(&18) C=C+1  
1040 IF SX=CHR$(&22) C=C+1  
1050 IF SX=CHR$(&28) C=C+1  
1060 IF SX=CHR$(&32) C=C+1  
1070 IF SX=CHR$(&38) C=C+1  
1080 RETURN

Using the same techniques, the information elements can be tested for correct content or stored in separate strings to be built into transmitted messages later.

Using the program fragment from above as an example, you can store the entire information element, including ID value and length in $A, by entering the following command:

165  $A = MIDS(SZ,P-1,L + 2)

This command would appear immediately after the information element length is moved into the numeric variable L, and the pointer P is pointing to the length byte.
The MID$ function enables you to copy a designation part of a string.

An explanation of the MID$ function in line 165 is:

a. In string Z, count from the beginning of the string to character number p - 1. For example, if the value of p is 10, count from the beginning of the string to the ninth character in the string.

b. Beginning with this character, copy the number of characters equal to L + 2 into string $A. Continuing with our example, assume that L = 15. Therefore, starting with the ninth character, copy the next 15 + 2 (17) characters into string $A.

In the programming example above, since P is pointing to the length octet, we start at p-1, the information element identifier. Next, as the identifier and length octets are not counted in the length value, we must transfer length + 2 octets.
CONVERTING PROGRAMS TO EXTENDABLE SIMP/L LAPD

The purpose of this section is to suggest some ways to upgrade non-extended SIMP/L LAPD programs to use the features available in Extendable SIMP/L LAPD.

Modifying program to include three of the Extendable SIMP/L LAPD features that are not available in non-extended SIMP/L LAPD are addressed in this section:

- Status control byte
- Multiple receive SAPIs and TEIs
- Frame status byte is added

Status Control Byte

If it is important to the test program to accept only responses with a SAPI and TEI that matches those transmitted in commands. In Extendable SIMP/L LAPD you can configure the Status Control Byte to a restricted address mode using the SET or SET CONFIG commands.

Most programs start with lines similar to these:

```
10 $A="MESSAGE TO BE BUILT INTO A PACKET"
20 SET SAPI=16
30 SET TEI=127
40 REM GO GET TEI ASSIGNMENT
50 GOSUB 500
60 SET TLI=A
70 SLO
```

In this program fragment, you could add a line 15 to use the SET command for address restriction:

```
15 SET RESTRICT
```

the configuration byte is now set by the program. Refer to the SET {fnctn} and SET CONFIG commands for more information.
Multiple Receive SAPIs and TEIs

It is a frequent practice to have a skeleton program with standardized setup procedures which can be added to, or merged with a particular test. For these programs, the SAPI and TEI are set to a fixed value by keyboard input or automatic assignment routine.

When this is the case, they are always set in the same line number by the same constant or variable. This means that the assignment of the receive values can be included in an update program which is then merged with each test program.

If, for example, you have a number of programs which all start the same as the above program fragment, you would create the program below:

```
15 SET RESTRICT
25 SET RSAPIO=16
35 SET RTEIO=127
65 SET RTEIO=A
```

This update can be saved to disk and MERGED into the desired programs. Refer to the MERGE command in Section 3.5 for more information.

If programs were developed without a skeleton framework, use the edit function to change line numbers, so that new lines are created, leaving the original lines unchanged.

For example, your program contains the following lines:

```
100 PRINT "INPUT TEI TO USE IN THIS TEST";
110 INPUT T
120 SET SAPI=16
130 SET TEI=T
140 SLON
150 IF LINKSTAT=1 GOTO 150
160 IF LINKSTAT=4 GOTO 190
170 PRINT "THE LINK CANNOT BE ESTABLISHED."
180 STOP
190 BUILD PRODIS,CRLEN,CREF,METYP,$A,$C,$D
```

You can edit line 120 to create a new line 125 to read:

```
125 SET RSAPIO=16
```
The steps are as follows (the * represents the cursor position):

!EDIT 120
120 *
12* Type 5 to set the new line number.
125* There is now a line 125 in the edit buffer which is identical to line 120. Press the right arrow five times.
125 SET *
125 SET * Press CTRL I to insert a blank.
125 SET R* Press the right arrow 5 times and press CTRL I to insert a blank.
125 SET RSAPI * Press the left arrow to move the cursor back to the new blank and type the letter R.
125 SET RSAPI0* Press the left arrow to move the cursor back to the new blank and type the number 0.
125 SET RSAPI0=16* Press CTRL P to display the rest of the line in the edit buffer.
125 SET RSAPI0=16* Press RETURN to enter the new line into the program.

Refer to the EDIT command in Section 3.5 for more information about editing BASIC commands.

In the preceding example, it would have been just as easy to type in the new line, but time would have been saved if a calculation was made, such as in the following:

120 IF LINKSTAT=0 SET SAPI=63

Once line 125 is entered, each time the program sets the SAPI value, it is a simple matter to edit line 125, change the line number, CTRL P to the end of the line, and put in the new value.
Frame Status Byte is Added

In Extendable SIMP/L LAPD received data packets are passed to your program with a frame status byte prefix which contains the following information:

- Type of frame received
- SAP and TEI used
- Poll/Final bit
- C/R bit

If you are converting a non-Extendable SIMP/L LAPD program to Extendable SIMP/L LAPD, you have several options available to you for handling the frame status byte:

- Use the UNEXTEND command at the beginning of your program to invoke the unextended version of Extendable SIMP/L LAPD. This will cause the frame status byte to be hidden and therefore require no additional modification to your program.

- If you want to access the frame status byte you must stay in EXTEND mode of Extendable SIMP/L LAPD.

When using the BREAK command, you must define a mnemonic that breaks the frame status byte from the beginning of the packet.

For example, to define a mnemonic for the frame status byte, you could use the following command:

```
DEFINE "FRST" = 8
```

Note

You cannot use the word FRSTAT as the mnemonic, since this is a SIMP/L LAPD variable.

Once the mnemonic is defined, it can be used with the BREAK command to break the frame status byte as the first element. For example:

```
BREAK FRST, $A
```

- You can also access the frame status byte using the FRSTAT read-only variable. Using FRSTAT is easier than using the BREAK command when you need to extract only the frame status byte. Refer to FRSTAT on page 5.6-14 for a description of the variable and an interpretation of the frame status.
5.7 MULTI-LINK SIMP/L LAPD

Introduction

Multi-Link SIMP/L LAPD supports a total of 64 logical links. To access the SIMP/L Multi-Link Simulator, select the SM MLAPD application from the Simulation window of the Applications Selection Menu. If you do not know how to configure the Chameleon for simulation or select a simulator, refer to Chapter 1 of this manual.

TGI Byte

Multi-Link SIMP/L LAPD includes the ability to use the TGI (Terminal Group Identifier) address byte. (If used, the TGI is the third byte of the LAPD Address field.) The SET TGI command assigns a TGI value to the currently selected link in the range 0 - 14. The TGI byte is handled as follows:

- If a valid TGI value (0 - 14) is assigned to a link, the TGI byte is used.
- If a TGI value > 14 is assigned to a link, the TGI byte is not used.
- If an invalid TGI value is assigned to a link (0 or 14, depending on LAPD implementation), it enables you to test the recovery of the Device Under Test to an invalid TGI value.

Link Selection

Multi-Link SIMP/L LAPD includes commands and variables which enable you to control the use of 64 logical links. Each of the 64 links is referred to by a unique link number in the range 0 - 63. Each of the 64 logical links has its own SAPI and TEI value, which are assigned as follows:

1. Select one of the 64 links (0 - 63) using the SET LINK command. For example: SET LINK = 1.
   Note that all links default to state 9, disabled.

2. Assign the link a SAPI value using the SET SAPI command. For example: SET SAPI = 16.

3. Assign the link a TEI value using the SET TEI command. For example: SET TEI = 127.

4. If applicable, Assign the link a TGI value using the SET TGI command. For example: SET TGI = 1.

5. When you select a link using SET LINK, you can then use the SIMP/L commands (shown on page 5.7-3) to set the link on (SLON), set the link off (SLOF), and transmit messages.
General Notes

If two or more links have the same address (SAPI:TEI or SAPI:TEI TGI combination), received frames will be considered to belong to the highest link number matching that address.

A link is disabled by selecting the link and setting the SAPI or TEI to an invalid value. You should ensure that the link is in the disconnected state before you disable it. If a link is disabled while in a connected (multi-frame) state, the device under test will see it as a Layer 1 failure. This can be useful for testing recovery procedures.

The state of the selected link can be determined using the LNKSTAT command. The STATUS command also returns the state of the selected link as part of its display. The STATE command displays the states of all 64 links.

Setting the SAPI and/or TEI value to an invalid value sets the link to the disabled state (9). This provides an easy means of testing lost link recovery and providing a means of ignoring unused links.

Two read-only variables are provided in Multi-Link LAPD to simplify link selection. FRELNK returns the number (0-63) of the lowest numbered disabled link, or -1 if no links are disabled. This enables you to select the next available disabled link, as follows:

\[
\text{SET LINK} = \text{FRELNK}
\]

RECLNK returns the link number of the last received message. This enables you to transmit a response to the last received message without knowing the link number. For example:

\[
\text{SET LINK} = \text{RECLNK}
\]
\[
\text{TRAN}
\]

The use of the Multi-Link commands and variables is demonstrated in the sample program on page 5.7-15.

Flow Control

Flow control occurs at the frame level, and depends on the amount of buffer space available on the Chameleon to receive data. If the receive buffer becomes full, the Chameleon declares the station busy and sends an RNR to the device under test. When the buffer is cleared, the Chameleon sends an RR to indicate that the busy condition has cleared.

To keep the receive buffer clear and avoid a station busy condition from disrupting your test, it is recommended that your program include a number of REC commands.
Multi-Link SIMP/L LAPD includes all common SIMP/L commands described in Section 5.3. It also includes the SIMP/L LAPD commands listed in Figure 5.7-1 below. The commands and variables specific to Multi-Link simulation are described beginning on the next page.

The following SIMP L LAPD commands are available in Multi-Link SIMP L LAPD without modification. The page number which describes the command in detail is provided for your reference.

<table>
<thead>
<tr>
<th>COMMAND/ VARIABLE</th>
<th>PAGE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNKSTAT</td>
<td>5.6-15</td>
<td>Returns the status of the selected link. Note the addition of state 9 (disabled) which is valid in Multi-Link SIMP L LAPD.</td>
</tr>
<tr>
<td>SET N200</td>
<td>5.6-18</td>
<td>Sets the value of N200 (maximum number of frame retransmissions) in the range 1 - 255</td>
</tr>
<tr>
<td>SET N201</td>
<td>5.6-18</td>
<td>Sets the value of N201 (maximum number of bytes in a frame) in the range 2 - 512</td>
</tr>
<tr>
<td>SET T200</td>
<td>5.6-19</td>
<td>Sets the value of the T200 timer in the range 0 - 255</td>
</tr>
<tr>
<td>SET T203</td>
<td>5.6-19</td>
<td>Sets the value of the T203 timer in the range 0 - 255</td>
</tr>
<tr>
<td>SET WINDOW</td>
<td>5.6-20</td>
<td>Sets the window size in the range 1 - 7</td>
</tr>
<tr>
<td>SET CONFIG</td>
<td>5.6-22</td>
<td>Sets the control configuration byte</td>
</tr>
<tr>
<td>TRUI</td>
<td>5.6-29</td>
<td>Transmits an unnumbered I-Frame</td>
</tr>
<tr>
<td>TRXIDC</td>
<td>5.6-30</td>
<td>Transmits an XID command frame</td>
</tr>
<tr>
<td>TRXIDR</td>
<td>5.6-31</td>
<td>Transmits an XID response frame</td>
</tr>
</tbody>
</table>

Figure 5.7-1: SIMP/L LAPD Commands Available in Multi-Link
SIMP/L Multi-Link

Commands

Figure 5.7-2 lists the commands and variables that are specific to Multi-Link SIMP/L LAPD. They are described in detail on the indicated pages.

<table>
<thead>
<tr>
<th>COMMAND/VARIABLE</th>
<th>PAGE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRELNK</td>
<td>5.7-5</td>
<td>Returns the number (0 - 63) of the lowest disabled link</td>
</tr>
<tr>
<td>FRSTAT</td>
<td>5.7-6</td>
<td>Returns the frame status byte. This variable is also available in SIMP/L LAPD, but the byte has a special Multi-Link interpretation</td>
</tr>
<tr>
<td>RECLNK</td>
<td>5.7-7</td>
<td>Returns the link number (0 - 63) of the last received message</td>
</tr>
<tr>
<td>SET LINK</td>
<td>5.7-8</td>
<td>Selects the link (0 - 63) under user control</td>
</tr>
<tr>
<td>SET SAPI</td>
<td>5.7-9</td>
<td>Assigns a SAPI value to the link under user control</td>
</tr>
<tr>
<td>SET TEI</td>
<td>5.7-10</td>
<td>Assigns a TEI value to the link under user control</td>
</tr>
<tr>
<td>SET TGI</td>
<td>5.7-11</td>
<td>Assigns a TGI value to the link under user control</td>
</tr>
<tr>
<td>STATE</td>
<td>5.7-12</td>
<td>Displays the states of all 64 links</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.7-13</td>
<td>Displays the status of the link under user control</td>
</tr>
<tr>
<td>TPRINT</td>
<td>5.7-14</td>
<td>Displays the contents of the trace buffer. This command is also available in SIMP/L LAPD, but has a special Multi-Link interpretation</td>
</tr>
</tbody>
</table>

Figure 5.7-2: Multi-Link SIMP/L Commands and Variables
## FRELNK

<table>
<thead>
<tr>
<th>Description</th>
<th>FRELNK is a read-only variable that returns the number of the lowest disabled link (0 - 63). If FRELNK returns -1, there are no disabled links.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Read-only Variable</td>
</tr>
<tr>
<td>Syntax</td>
<td>See examples below</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>None</td>
</tr>
</tbody>
</table>
| Example     | PRINT FRELNK  
IF FRELNK...  
SET LINK = FRELNK                                                                                                          |
FRSTAT

Description
FRSTAT is a read-only variable that returns a 2-byte value. The first byte contains the link number. The second byte is interpreted as shown in the diagram below.

![Diagram of FRSTAT interpretation](image)

- **High bit low bit**
  - 00 = UI frame
  - 01 = XID frame
  - 10 = I-frame
  - 11 = CMDR (Diagnostic field)
- **Reserved**
  - 0 = Command frame
  - 1 = Response frame
- **Poll Final bit**
  - 0 = Poll Final bit clear
  - 1 = Poll Final bit set

Figure 5.7-3: Multi-Link Frame Status Byte Interpretation (Second Byte)

<table>
<thead>
<tr>
<th>Type</th>
<th>Read-only variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>(See example below.)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>FRS.</td>
</tr>
<tr>
<td>See Also</td>
<td>RECLNK</td>
</tr>
<tr>
<td>Example</td>
<td>PRINT FRSTAT</td>
</tr>
</tbody>
</table>
# RECLNK

**Description**  
RECLNK is a read-only variable that returns the number of the link from which the last data was received. It returns one byte.

**Type**  
Read-only Variable

**Syntax**  
See examples below

**Abbreviation**  
None

**Example**  
```
PRINT RECLNK
IF RECLNK....
SET LINK = RECLNK
```
SET LINK

Description
The SET LINK command places one of the 64 available links under user control. You can then use SIMP L commands to set the link on or off, or transmit messages over the link.

Type
Statement or Direct

Syntax
SET LINK = exp

where exp is any valid expression with a value in the range 0 - 63.

Abbreviation
None

See Also
STATUS, FRELNK, RECLNK

Examples
SET LINK = A
SET LINK = 7
SET LINK = 2 * B
SET LINK = FRELNK
SET LINK = RECLNK
SET SAPI

Description  The SET SAPI command sets the Service Access Point Identifier (SAPI) value for the selected link. Normal values are:

- 0  Call Control procedures
- 16 Packet communication procedures
- 63  Management procedures

Type  Statement or Direct

Syntax  SET SAPI = x

where x is the SAPI value in the range 0 - 63.

To disable the selected link (STATE = 9), set the SAPI to a value outside the range. Normally the link should be in the disconnected state before you disable it.

Abbreviation  None

See Also  SET LINK, SET TEI, SET TGI

Example  SET SAPI = 0
## SET TEI

**Description**

The SET TEI command sets the Terminal Endpoint Identifier (TEI) value for the selected link. Normal values are:

- **0 - 63**: Non-Automatically assigned values
- **64 - 126**: Automatically assigned values
- **127**: Broadcast value

**Type**

Statement or Direct

**Syntax**

SET TEI = x

where x is the TEI value in the range 0 - 127.

To disable the selected link (STATE = 9), set the TEI to a value outside the range. Normally the link should be in the disconnected state before you disable it.

**Abbreviation**

None

**See Also**

SET LINK, SET TGI, SET SAPI

**Example**

SET TEI = 0
SET TGI

Description
The SET TGI command sets the Terminal Group Identifier (TGI) address byte for the selected link. The simulator handles the TGI byte as follows:

- If a valid TGI value (1 - 14) is assigned to a link, the TGI byte is used.
- If a value >15 is assigned to a link, the TGI byte is not used.
- If an invalid TGI value is assigned to a link (0 or 14, depending on LAPD implementation), it enables you to test the recovery of the Device Under Test to an invalid TGI value.

Type
Statement or Direct

Syntax
SET TGI = x

where x is the TGI value in the range 0 - 255, as described above. A value greater than 14 does not disable the link; it causes the selected link to transmit and accept frames with a 2-byte (SAPI/TEI) address field.

Abbreviation
None

See Also
SET LINK, SET TEI, SET SAPI

Example
SET TGI = 1
STATE

Description

STATE displays the state of all 64 links in a 4 x 16 matrix corresponding to the link number. The STATE display uses the following format:

<table>
<thead>
<tr>
<th>LINK</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>1</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>2</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>3</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>4</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>5</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>6</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>7</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>8</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>9</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>10</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>11</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>12</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>13</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>14</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>15</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

The STATE values are as follows:

<table>
<thead>
<tr>
<th>STATE</th>
<th>LINK STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LinkDisconnected</td>
</tr>
<tr>
<td>1</td>
<td>LinkConnectionRequested</td>
</tr>
<tr>
<td>2</td>
<td>FrameRejected</td>
</tr>
<tr>
<td>3</td>
<td>DisconnectRequested</td>
</tr>
<tr>
<td>4</td>
<td>InformationTransfer</td>
</tr>
<tr>
<td>5</td>
<td>LocalStationBusy</td>
</tr>
<tr>
<td>6</td>
<td>RemoteStationBusy</td>
</tr>
<tr>
<td>7</td>
<td>Local and RemoteStationBusy</td>
</tr>
<tr>
<td>8</td>
<td>RemoteStationnotResponding</td>
</tr>
<tr>
<td>9</td>
<td>LinkDisabled</td>
</tr>
</tbody>
</table>

Figure 5.7-4: Multi-Link SIMP/L LAPD Link States

Type

Statement or Direct

Syntax

STATE

Abbreviation

None

Example

STATE
## STATUS

<table>
<thead>
<tr>
<th>Description</th>
<th>The STATUS command displays the following information about the currently selected link:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Link number</td>
</tr>
<tr>
<td></td>
<td>• SAPI, TEI, and TGI values</td>
</tr>
<tr>
<td></td>
<td>• State</td>
</tr>
<tr>
<td></td>
<td>• T200 and T203 timers</td>
</tr>
<tr>
<td></td>
<td>• N201</td>
</tr>
<tr>
<td></td>
<td>• N200</td>
</tr>
<tr>
<td></td>
<td>• Window</td>
</tr>
<tr>
<td></td>
<td>• Modulo</td>
</tr>
<tr>
<td></td>
<td>• Network or Subscriber</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Statement or Direct</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>STATUS</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>STA.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>In Multi-Link SIMP/L LAPD, the STATUS display is in the following format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>!STATUS</td>
</tr>
<tr>
<td></td>
<td>LINK  SAPI  TEI  TGI  STATE</td>
</tr>
<tr>
<td>0</td>
<td>16    127   1   4</td>
</tr>
<tr>
<td>Timer T200 Value - 10</td>
<td></td>
</tr>
<tr>
<td>Timer T203 Value - 20</td>
<td></td>
</tr>
<tr>
<td>Maximum frame size (N201) - 260</td>
<td></td>
</tr>
<tr>
<td>Number of Re-transmissions (N200) - 3</td>
<td></td>
</tr>
<tr>
<td>Window - 3 MOD128</td>
<td></td>
</tr>
<tr>
<td>Simulating a network</td>
<td></td>
</tr>
</tbody>
</table>

The top line displays the link number, SAPI value, TEI value, TGI value, and state of the link currently under user control. (If the TGI address byte is not being used, no value will be displayed for TGI.) Refer to page 5.7-12 for a list of STATE values and their meanings.

A blank space under the SAPI or TEI heading indicates that an invalid (disabling) value has been selected for that link.
TPRINT

Description

The TPRINT command displays the contents of the trace buffer. For Multi-Link SIMPL LAPD, the TPRINT display includes a 2-byte frame status for received frames. In Extendable SIMPL, frame status is a single byte. The second byte in Multi-Link is used to indicate the link number.

Since Multi-Link frame status is two bytes, you must take this into account if you are adapting an existing Extendable SIMPL LAPD program to run on Multi-Link SIMPL.

The recommended procedure is to redefine the mnemonic used to break the frame status byte to be 16 bits long. This eliminates the need to modify each BREAK statement that you may have in existing SIMPL LAPD programs.

Example:

```
DEFINE FRST = 8  (Defines mnemonic to break the 1-byte frame status in Extendable SIMPL LAPD)
```

change to

```
DEFINE FRST = 16 (Defines mnemonic to break the 2-byte frame status in Multi-Link SIMPL LAPD)
```

Type

Statement or Direct

Syntax

TPRINT

Example

TPRINT

The format of the TPRINT display for Multi-Link SIMPL is:

```
T = Transmitted frame
R = Received frame
00:00:00:00:00 = Time stamp
0041 FRSTAT = Frame status (2 bytes - received frame only)
LEN 1234 08 01 A3 05 04 88 90 80 = Contents of received data packet in hex
```
Sample Program

The sample below illustrates a typical startup procedure for Multi-Link simulation.

10 SET LINK = FRELNK
   Puts the lowest numbered disabled link under user control.

15 SET TEI = 126
   Assigns selected link the TEI of 126.

20 SET SAPI = 16
   Assigns selected link a SAPI = 16.

25 SET TGI = 2
   Assigns selected link the TGI of 2. If the TGI byte is not being used, omit this line.

30 SLON
   Sets the link on.

40 IF LNKSTAT = 1, GOTO 40
   Waits while link request is being processed.

50 IF LNKSTAT = 4 GOTO 80
   If link enters information transfer state, jumps to line 80 to build and transmit a message.

60 PRINT "CAN'T ESTABLISH LINK"
   If status of link is other than Connect Requested or information transfer, displays message.

70 STOP
   Stops program if link cannot be established.

80 $A = "hello"
   Assigns a string to string variable $A.

90 $B = "world"
   Assigns a string to string variable $B.

100 BUILD $A, $B
   If link is in information transfer, builds a transmission message from string variables A and B.

110 TRAN
   Transmits the message.

120 REC
   Attempts to receive a message.

130 IF LENGTH = 0 GOTO 120
   Loops until a message is received.

140 SET LINK = RECLNK
   Selects the link from which the message was received.

150 BUILD "MESSAGE RESPONSE"
   Builds a response message to the selected link.

160 GOTO 110
   Loops to transmit the response message and wait for another message.
5.8 SIMP/L V.120

Introduction

SIMP/L V.120 provides 64 independent links and adheres to the following V.120 protocol requirements:

- The C/R bit is set to 0 for all commands, and to 1 for all responses, regardless of the sending station
- I-Frames can be command or response frames
- When a command reject (CMDR) occurs the link is automatically restarted

To access the SIMP/L V.120 Simulator, select the SM_V120 application from the Simulation window of the Applications Selection Menu. If you do not know how to configure the Chameleon for simulation or select a simulator, refer to Chapter 1 of this manual.

V.120 Address

The format of the V.120 address field can be viewed as a single 13-bit Logical Link Identifier (LLI) field or as two separate fields (LLI0 and LLI1). This is shown as follows:

```
<table>
<thead>
<tr>
<th>bits</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 7 6 5 4 3 2 1</td>
<td>Octet 2 (Address Octet 1)</td>
</tr>
<tr>
<td>LLI0</td>
<td>C/R  E A 0</td>
</tr>
<tr>
<td></td>
<td>Octet 2 (Address Octet 1)</td>
</tr>
<tr>
<td>LLI1</td>
<td>E A 1</td>
</tr>
</tbody>
</table>
```

The LLI0 is the high order 6 bits of the LLI. The LLI1 is the low order 7 bits of the LLI. The LLI is a concatenation of the LLI0 field with the LLI1 field. The LLI can take on values in the range 0 - 8191, with the following reserved values:

<table>
<thead>
<tr>
<th>LLI (In Decimal)</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>In-channel signaling</td>
</tr>
<tr>
<td>1 - 255</td>
<td>Reserved for future standardization</td>
</tr>
<tr>
<td>256</td>
<td>Default LLI</td>
</tr>
<tr>
<td>257-2047</td>
<td>For LLI assignment</td>
</tr>
<tr>
<td>2048-8190</td>
<td>Reserved for future standardization</td>
</tr>
<tr>
<td>8191</td>
<td>In-channel layer management</td>
</tr>
</tbody>
</table>

Figure 5.8-1: LLI Values
EA0 is the octet 2 address extension bit, which is set to 0. EA1 is the octet 3 address extension bit, which is set to 1 for two octet address field.

With SIMP/L V.120, you select one of the 64 links (0-63) using the SET LINK command. You then assign the link an LLI with the SET LLI command as a single decimal value as shown in Figure 5.8-1 on the previous page. All links default to state 9, disabled.

When you select a link using SET LINK, you can then use the SIMP/L commands (shown on the next page) to set the link on (SLON), set the link off (SLOF), and transmit messages.

A link is disabled by selecting the link and setting the LLI to an invalid value. You should ensure that the link is in the disconnected state before you disable it. If a link is disabled while in a connected (multi-frame) state, the device under test will see it as a Layer 1 failure. This can be useful for testing recovery procedures.

The state of the selected link can be determined using the LNKSTAT command. The STATUS command also returns the state of the selected link as part of its display. The STATE command displays the states of all 64 links.

Two read-only variables are provided in V.120. FRELNK returns the number (0-63) of the lowest numbered disabled link, or -1 if no links are disabled. This enables you to select the next available disabled link, as follows:

\[
\text{SET LINK} = \text{FRELNK}
\]

RECLNK returns the link number of the last received message. This enables you to transmit a response to the last received message without knowing the link number. For example:

\[
\text{SET LINK} = \text{RECLNK} \\
\text{TRAN}
\]

In V.120, sequenced I-Frames can be either commands or responses. To determine the C/R bit of received frames, use the FRSTAT variable, which returns a frame status byte indicating the C/R bit value. Response I-frames can be transmitted using the command RTRAN.

The use of the V.120 commands and variables is demonstrated in the sample program on page 5.8-13.
Flow Control

Flow control occurs at the frame level, and depends on the amount of buffer space available on the Chameleon to receive data. If the receive buffer becomes full, the Chameleon declares the station busy and sends an RNR to the device under test. When the buffer is cleared, the Chameleon sends an RR to indicate that the busy condition has cleared.

To keep the receive buffer clear and avoid a station busy condition from disrupting your test, it is recommended that your program include a number of REC commands.

Commands

SIMP/L V.120 includes all common SIMP/L commands described in Section 5.3. It also includes the SIMP/L LAPD commands listed in Figure 5.8-2 below. The commands and variables specific to V.120 simulation are described beginning on the next page.

SIMP/L LAPD Commands

The following SIMP/L LAPD commands are available in SIMP/L V.120 without modification. The page number which describes the command in detail is provided for your reference.

<table>
<thead>
<tr>
<th>COMMAND/ VARIABLE</th>
<th>PAGE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNKSTAT</td>
<td>5.6-15</td>
<td>Returns the status of the selected link</td>
</tr>
<tr>
<td>SET N200</td>
<td>5.6-18</td>
<td>Sets the value of N200 (maximum number of frame retransmissions) in the range 1 - 255</td>
</tr>
<tr>
<td>SET N201</td>
<td>5.6-18</td>
<td>Sets the value of N201 (maximum number of bytes in a frame) in the range 2 - 512</td>
</tr>
<tr>
<td>SET T200</td>
<td>5.6-19</td>
<td>Sets the value of the T200 timer in the range 0 - 255</td>
</tr>
<tr>
<td>SET T203</td>
<td>5.6-19</td>
<td>Sets the value of the T203 timer in the range 0 - 255</td>
</tr>
<tr>
<td>SET WINDOW</td>
<td>5.6-20</td>
<td>Sets the window size in the range 1-7</td>
</tr>
<tr>
<td>SET CONFIG</td>
<td>5.6-22</td>
<td>Sets the control configuration byte</td>
</tr>
<tr>
<td>TRUI</td>
<td>5.6-29</td>
<td>Transmits an unnumbered l-Frame</td>
</tr>
<tr>
<td>TRXIDC</td>
<td>5.6-30</td>
<td>Transmits an XID command frame</td>
</tr>
<tr>
<td>TRXIDR</td>
<td>5.6-31</td>
<td>Transmits an XID response frame</td>
</tr>
</tbody>
</table>

Figure 5.8-2: SIMP/L LAPD Commands Available in V.120 LAPD
SIMP/L V.120 Commands

Figure 5.8-3 lists the commands and variables that are specific to SIMP/L V.120. They are described in detail on the indicated pages.

<table>
<thead>
<tr>
<th>COMMAND/ VARIABLE</th>
<th>PAGE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRELNK</td>
<td>5.8-5</td>
<td>Returns the number (0 - 63) of the lowest disabled link</td>
</tr>
<tr>
<td>FRSTAT</td>
<td>5.8-6</td>
<td>Returns the frame status byte. This variable is also available in SIMP/L LAPD, but the byte has a V.120-specific interpretation</td>
</tr>
<tr>
<td>RECLNK</td>
<td>5.8-7</td>
<td>Returns the link number (0 - 63) of the last received message</td>
</tr>
<tr>
<td>RTRAN</td>
<td>5.8-8</td>
<td>Transmits an I-Frame response</td>
</tr>
<tr>
<td>SET LINK</td>
<td>5.8-9</td>
<td>Selects the link (0 - 63) under user control</td>
</tr>
<tr>
<td>SET LLI</td>
<td>5.8-10</td>
<td>Sets the Logical Link Identifier of the link under user control</td>
</tr>
<tr>
<td>STATE</td>
<td>5.8-11</td>
<td>Displays the state of all 64 links</td>
</tr>
<tr>
<td>STATUS</td>
<td>5.8-12</td>
<td>Displays the status of the link under user control</td>
</tr>
</tbody>
</table>

Figure 5.8-3: V.120 SIMP/L Commands and Variables
FRELNK

Description
FRELNK is a read-only variable that returns the number of the lowest disabled link (0 - 63). If FRELNK returns -1, there are no disabled links.

Type
Read-only Variable

Syntax
See examples below

Abbreviation
None

Example
PRINT FRELNK
IF FRELNK....
SET LINK = FRELNK
FRSTAT

Description

FRSTAT is a read-only variable that returns a 2-byte value. The first byte contains the link number. The second byte is interpreted as shown in the diagram below.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>High bit</td>
<td>low bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>UI frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>XID frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>I-frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>CMDR (Diagnostic field)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Command frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Response frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Poll/Final bit clear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Poll/Final bit set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.8-4: V.120 Frame Status Byte Interpretation (Second Byte)

Type

Read-only variable

Syntax

(See example below.)

Abbreviation

FRS.

See Also

RECLNK, FRSTAT

Example

PRINT FRSTAT
RECLNK

Description
RECLNK is a read-only variable that returns the link number (0 - 63) of the last received message.

Type
Read-only Variable

Syntax
See examples below

Abbreviation
None

Example
PRINT RECLNK
IF RECLNK
SET LINK = RECLNK
### RTRAN

**Description**  
The RTRAN command transmits an I-Frame response. It sets the C/R bit to the response value and transmits the frame.

Sequenced I-Frames can be commands or responses. Received frame C/R bits can be determined using the FRSTAT variables which returns the frame status byte. In V.120 the C/R bit is set as follows:

- Command frame C/R bit is set to 0
- Response frame C/R bit is set to 1

**Type**  
Statement or Direct

**Syntax**  
RTRAN

**Abbreviation**  
None

**Example**  
RTRAN
## SET LINK

**Description**
The SET LINK command places one of the 64 available links under user control. You can then use SIMP/L commands to set the link on or off, or transmit messages over the link.

**Type**
Statement or Direct

**Syntax**
SET LINK = exp

where `exp` is any valid expression with a value in the range 0 - 63.

**Abbreviation**
None

**See Also**
STATUS, FRELNK, RECLNK

**Examples**
SET LINK = A
SET LINK = 7
SET LINK = 2 * B
SET LINK = FRELNK
SET LINK = RECLNK
SET LLI

Description
The SET LLI command sets the Logical Link Identifier value for the selected link. The LLI specifies the layer two service type requested or supported, as shown in the table below.

Type
Statement or Direct

Syntax
SET LLI = x
where x is the Logical Link Identifier value, within the following range:

<table>
<thead>
<tr>
<th>LLI</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>In-channel signaling</td>
</tr>
<tr>
<td>1 - 255</td>
<td>Reserved for future standardization</td>
</tr>
<tr>
<td>256</td>
<td>Default LLI</td>
</tr>
<tr>
<td>257-2047</td>
<td>For LLI assignment</td>
</tr>
<tr>
<td>2048-8190</td>
<td>Reserved for future standardization</td>
</tr>
<tr>
<td>8191</td>
<td>In-channel layer management</td>
</tr>
</tbody>
</table>

To disable the selected link (STATE = 9), set the LLI to a value outside the range 0 - 8191. Normally the link should be in the disconnected state before you disable it.

Abbreviation
None

See Also
SET LINK

Example
SET LLI = 0
STATE

Description

STATE displays the state of all 64 links, using the following values:

<table>
<thead>
<tr>
<th>STATE</th>
<th>LINK STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link Disconnected</td>
</tr>
<tr>
<td>1</td>
<td>Link Connection Requested</td>
</tr>
<tr>
<td>2</td>
<td>Frame Rejected</td>
</tr>
<tr>
<td>3</td>
<td>Disconnect Requested</td>
</tr>
<tr>
<td>4</td>
<td>Information Transfer</td>
</tr>
<tr>
<td>5</td>
<td>Local Station Busy</td>
</tr>
<tr>
<td>6</td>
<td>Remote Station Busy</td>
</tr>
<tr>
<td>7</td>
<td>Local and Remote Station Busy</td>
</tr>
<tr>
<td>8</td>
<td>Remote Station not Responding</td>
</tr>
<tr>
<td>9</td>
<td>Link Disabled</td>
</tr>
</tbody>
</table>

Figure 5.8-5: V.120 Link States

Type

Statement or Direct

Syntax

STATE

Abbreviation

None

Example

STATE
STATUS

Description
The STATUS command displays the following information about the currently selected link:

- Link number
- LLI Value
- State
- T200 and T203 timers
- N201
- N200
- Window
- Modulo

Type
Statement or Direct

Syntax
STATUS

Abbreviation
STA.

Example
In SIMP/L V.120 LAPD, the STATUS command display is in the following format:

```
!STATUS
LINK     LLI     STATE
 3        147     4
Timer T200 Value - 10
Timer T203 Value - 20
Maximum frame size (N201) - 260
Number of Re-transmissions (N200) - 3
Window - 3 MOD128
```

Refer to page 5.8-10 for a list of LLI values and their meanings. Refer to page 5.8-11 for a list of STATE values and their meanings.

A blank space under the LLI heading indicates that an invalid (disabling) value has been selected for the LLI.
Sample Program

The sample below illustrates a typical startup procedure for V.120 simulation.

10 SET LINK = FRELNK
   Puts the lowest numbered disabled link under user control.

20 SET LLI = 0
   Assigns selected link the LLI of 0.

30 SLON
   Sets the link on.

40 IF LNKSTAT = 1, GOTO 40
   Waits while link request is being processed.

50 IF LNKSTAT = 4 GOTO 80
   If link enters information transfer state, jumps to line 80 to build and transmit a response.

60 PRINT "CAN'T ESTABLISH LINK"
   If status of link is other than Connect Requested or information transfer, displays message.

70 STOP
   Stops program if link cannot be established.

80 $A = "hello"
   Assigns a string value to string variable $A.

90 $B = "world"
   Assigns a string value to string variable $B.

100 BUILD $A, $B
   If link is in information transfer, builds a transmission message from string variables A and B.

110 TRAN
   Transmits the message.

120 REC
   Attempts to receive a message.

130 IF LENGTH = 0 GOTO 120
   Loops until a message is received.

140 SET LINK = RECLNK
   Selects the link from which the message was received.

150 BUILD " MESSAGE RESPONSE"
   Builds a response message to the selected link.

160 GOTO 110
   Loops to transmit the response message and wait for another message.
6.1 CHAMELEON BSC FUNDAMENTALS

Introduction

With Tekelec's Chameleon Binary Synchronous software, you can emulate and perform testing on a Bisynchronous device. Chameleon BSC is flexible, allowing you to deviate from standard protocols during hardware/software development and to compensate for protocol deviations in equipment supplied to you by others.

Chameleon BSC has special commands for transmission, reception, timing control, and pin level interrogation and setting. In addition, all of the Chameleon BASIC commands are available in BSC.

BSC Applications

You can use Chameleon BSC for the following types of tasks:

• Use Chameleon BSC control characters, data strings, and other features to derive and finish your own particular BSC protocol.

• Emulate a specific device or an entire network, enabling you to develop your own devices, networks, software and systems into a working model of a live environment.

• Perform positive testing by supplying your device or network with correct information to see if all procedures are working correctly.

• Perform negative testing by deliberately sending incorrect data in order to test error recovery and system/device susceptibility to failure induced by invalid data.

• Handle ACKs and NAKs, create multiple data blocks, transmit and receive data in a controlled environment, and perform block checks.
BSC Frame

The diagram below illustrates a sample BSC frame, with a brief description of its components.

```
SYN SYN STX data ETX BCC FF
```

A. Synchronizing Characters announce that data is arriving.
B. Start of Text indicates beginning of data.
C. Data
D. End of Text indicates end of the data.
E. Block Check Characters check accuracy of data transmission.
F. Padding characters (FF hex) that fill the line when data is not being transmitted.

Transmission

In Chameleon BSC, a transmission buffer is used to build a frame for transmission. There are three commands that affect the BSC transmission buffer.

The TXBUFFER command assigns the contents of the transmission buffer; that is, the data that will be transmitted. Refer to page 6.5-9 for more information.

The TXSTATUS command assigns a status control byte that defines how the contents of the transmission buffer will be transmitted. For example, the control status byte determines start and end framing characters, use of transparent or text mode, and whether a good or bad CRC is sent. Refer to page 6.5-10 for more information.

The TRAN command transmits the contents of the transmission buffer according to the status control byte. Refer to page 6.5-8 for more information.

Reception

The trace buffer gives you access to the frames received. The REC command transfers information from the acquisition buffer to the trace buffer. The TPRINT command displays the contents of the trace buffer.

Refer to Section 3.2 for more information about simulation buffers and their usage.
6.2 CHAMELEON BSC PARAMETER SET-UP MENU

Introduction This section describes the parameters and valid values for the BSC menu. This menu enables you to configure and save parameters for running BSC simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu (option S), refer to Section 2.3.

---

** BISYNC PARAMETER SET-UP **

A. - Simulate : DCE  
B. - Bit Rate : 9600  
C. - Code : EBCDIC  
D. - BCC : CRC-16  
E. - Sync 1 : 32  
F. - Sync 2 : 32  
G. - SOH : 01  
H. - STX : 02  
I. - ETX : 03  
J. - DLE : 10  
K. - EOT : 37  
L. - ENQ : 20  
M. - ETB : 26  
N. - ITB : 1F

Z = run BISYNC,  ? = Help,  S = Save parameters,  ESC = Main Menu

---

BSC Parameters The BSC parameter fields and legal values are as follows:

A. Simulate  DTE  DCE
B. Bit Rate  50 - 64000
C. Code  EBCDIC  
      ASCII/no parity  
      ASCII/even parity  
      ASCII/odd parity
D. BCC (Block Control Char.)  CRC-16
   CCITT

E. Sync 1  2-digit hex number
F. Sync 2  2-digit hex number
G. SOH (Start of Header)  2-digit hex number
H. STX (Start of Text)  2-digit hex number
I. ETX (End of Text)  2-digit hex number
J. DLE (Data Link Escape)  2-digit hex number
K. EOT (End of Transmission)  2-digit hex number
L. ENQ (Enquiry)  2-digit hex number
M. ETB (End of Text Block)  2-digit hex number
N. ITB (Intermediate Text Block)  2-digit hex number
6.3 CHAMELEON BSC MNEMONICS

Introduction

This section describes the default mnemonic table for Chameleon BSC. For a general description of the use of mnemonic tables, refer to Section 3.3.

Entries

The BSC mnemonic table has four columns:

- Mnemonic name (1-6 characters)
- Decimal value
- Hexadecimal value
- Binary value

Table 6.3-1 shows the default BSC mnemonic table. You can define and save additional BSC mnemonic tables using the BASIC mnemonic commands described in Section 3.4.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK0</td>
<td>112</td>
<td>70</td>
<td>01110000</td>
</tr>
<tr>
<td>ACK1</td>
<td>97</td>
<td>61</td>
<td>01100001</td>
</tr>
<tr>
<td>WABT</td>
<td>127</td>
<td>7F</td>
<td>01111111</td>
</tr>
<tr>
<td>SOH</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>STX</td>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
<tr>
<td>ETB</td>
<td>38</td>
<td>26</td>
<td>00100110</td>
</tr>
<tr>
<td>ETX</td>
<td>3</td>
<td>03</td>
<td>00000011</td>
</tr>
<tr>
<td>ITB</td>
<td>31</td>
<td>1F</td>
<td>00011111</td>
</tr>
<tr>
<td>EOT</td>
<td>55</td>
<td>37</td>
<td>00110111</td>
</tr>
<tr>
<td>ENQ</td>
<td>45</td>
<td>2D</td>
<td>00101101</td>
</tr>
<tr>
<td>DLE</td>
<td>16</td>
<td>10</td>
<td>00010000</td>
</tr>
<tr>
<td>SYN</td>
<td>50</td>
<td>32</td>
<td>00110010</td>
</tr>
<tr>
<td>ACK</td>
<td>46</td>
<td>2E</td>
<td>00101110</td>
</tr>
<tr>
<td>NAK</td>
<td>61</td>
<td>3D</td>
<td>00111101</td>
</tr>
</tbody>
</table>

Table 6.3-1: BSC Default Mnemonic Table
6.4 BSC COMMAND INDEX

In this section, the BSC commands are listed by function in two tables. This functional listing provides only the command name, a brief description, and the command type (Statement or Direct). For more information about a command, refer to the page number indicated.

Remember that you can also use the Chameleon BASIC commands that are described in Section 3.5.

**TABLE 6.4-1: BSC TRANSMISSION AND RECEPTION COMMANDS**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>REC</td>
<td>6.5-5</td>
<td>S or D</td>
<td>Receives a block or character.</td>
</tr>
<tr>
<td>RXLENGTH</td>
<td>6.5-6</td>
<td>Variable</td>
<td>Read-only variable that returns the length of the last received block.</td>
</tr>
<tr>
<td>TRAN</td>
<td>6.5-8</td>
<td>S or D</td>
<td>Transmits the contents of the transmission buffer.</td>
</tr>
<tr>
<td>TXBUFFER</td>
<td>6.5-9</td>
<td>S or D</td>
<td>Defines the contents of the transmission buffer (assembles data in the specified order to be transmitted).</td>
</tr>
<tr>
<td>TXSTATUS</td>
<td>6.5-10</td>
<td>S or D</td>
<td>Defines the control status byte that determines how the data in the transmission buffer will be transmitted.</td>
</tr>
</tbody>
</table>

**TABLE 6.4-2: BSC MISCELLANEOUS COMMANDS**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC16</td>
<td>6.5-2</td>
<td>S or D</td>
<td>Calculates the two-byte CRC of a string and stores the result in another string.</td>
</tr>
<tr>
<td>IDLE</td>
<td>6.5-3</td>
<td>S or D</td>
<td>Transmits continuous SYNs or FFs to mark the line when not transmitting data.</td>
</tr>
<tr>
<td>LRC</td>
<td>6.5-4</td>
<td>S or D</td>
<td>Calculates the one-byte LRC of a string and stores the result in another string.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>6.5-7</td>
<td>S or D</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
</tbody>
</table>
6.5 BSC COMMANDS

Introduction

This section provides a complete description of the BSC commands. The commands are organized alphabetically, one command per page.

If you know the name of the command you want to use, you can look up the command in this section. If you are unsure of the command you need, refer to the previous section which lists the commands by function.

The following information is provided in this section:

- Command description
- Type of command (Statement or Direct)
- Syntax
- Command abbreviation
- Related commands (See Also)
- Example(s) of command usage
CRC16

Description
This function calculates the two byte polynomial CRC (Cyclic Redundancy Check) characters for a string of data. The CRC polynomial is:

\[ x^{16} + x^{15} + x^2 + 1 \]

Type
Statement or Direct

Syntax
CRC16($A)

where: $A$ is the data.

Abbreviation
None

See Also
LRC, TXSTATUS, REC, LEFT$

Example 1
In this example, the two CRC bytes are extracted from a received frame and checked for accuracy. This assumes that the following string is received:

$A = \text{HEX} > 02C1C2C3C4C5C6 03 \text{ba 93 FF}$

10 REC $A$
20 IF RXLENGTH=0 GOTO 10
30 DISP $F$
40 $B$=RIGHTS($A, 3$)
50 $B$=LEFTS($B, 2$)
60 $C$=MIDS($A, 2, \text{LEN}(A)-3$)
70 $D$=CRC16($C$)
80 IF $D$=$B$ PRINT "GOOD CRC"
90 PRINT "BAD CRC"

!RUN <RETURN>

Receives a frame.
If no frame received, loops to try again.
Displays the received frame.
Extracts the 3 rightmost characters of the frame.
Extracts the 2 leftmost characters of $B$ (CRC).
Stores the frame (without the CRC bytes and STX) to $C$.
Calculates the CRC of $C$ and stores it in $D$.
If calculated CRC matches extracted CRC, prints message.
If calculated CRC does not match extracted CRC, prints message.
### IDLE

**Description**

The IDLE command determines what is transmitted when the line is idle. The default condition is for the Chameleon to mark the line by sending continuous FFs.

The IDLE command may not be used while data is being transmitted, so be sure to allow time for outstanding transmissions to be cleared before using it.

**Type**

Statement or Direct

**Syntax**

<table>
<thead>
<tr>
<th>IDLE = SYNC</th>
<th>Idles the line by sending continuous SYNs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE = MARK</td>
<td>Idles the line by sending continuous FFs.</td>
</tr>
</tbody>
</table>

**Abbreviation**

ID.

**Example**

This example causes the Data LEDs on the Chameleon to flash continuously as the idle character changes between the SYN and FF.

```
10 IDLE=SYNC
20 IDLE=MARK
30 GOTO 10
! RUN <RETURN>
```
**LRC**

**Description**
This function calculates the LRC (Longitudinal Redundancy Check) for a string of data. The LRC polynomial is:

\[ x^8 + 1 \]

**Type**
Statement or Direct

**Syntax**
LRC($A)

*where: $A is the data.*

**Abbreviation**
None

**See Also**
CRC16

**Example**
In this example, the LRC byte is extracted from a received frame and checked for accuracy. This assumes that the following string is received:

$A = \text{HEX} > 02C1C2C3C4C5C6 03 \text{ ba } 93 \text{ FF}

```
10 REC $A
20 IF RXLENGTH=0 GOTO 10
30 DISP $A
40 $B=RIGHTS($A,2)
50 $B=LEFT$(SB.1)
60 $C=MIO$(SA.2.LEN($A)-2)
70 $D=LRC($C)
80 IF $D=$B PRINT "GOOD LRC"
90 PRINT "BAD LRC"

!RUN <RETURN>
```

Receives a frame.
If no frame received, loops to try again.
Displays the received frame.
Extracts the 2 rightmost characters of the frame.
Extracts the leftmost character of $B (LRC).
Stores the frame (without the LRC byte and STX) to $C.
Calculates the LRC of $C and stores it in $D.
If calculated LRC matches extracted LRC, prints message.
If calculated LRC does not match extracted LRC, prints message.
REC

Description
REC takes the next received block in sequence from the buffer and assigns it to one or more string variables. A string variable can be assigned a maximum of 255 characters. When the 255-character maximum is reached, data is stored in the next string variable indicated in the command.

If string variables included in the REC command are not needed for reception, the previous contents of the string are unaffected.

Refer to Section 3.2 for a description of the Simulation buffers. Remember to FLUSH the reception buffer before receiving and storing traffic, if necessary.

Type
Statement or Direct

Syntax
REC $A, $B...

Abbreviation
None

See Also
TPRINT, DISPF

Example
This example waits to receive a frame and then displays it.

```
10 REC $A
20 IF RXLENGTH=0 GOTO 10
30 DISPF
!RUN <RETURN>
```

Receives a frame.
If no frame received, loops to try again.
Displays the received frame.
RXLENGTH

Description  RXLENGTH is a read-only variable that returns the length of the last received block. If the value returned is 0, no data was received.

Type  Variable

Syntax  (See example below.)

Abbreviation  RX.

See Also  REC

Example  This example waits until a block is received and then prints the length of the block.

```
10 REC $A
20 IF RXLENGTH = 0 GOTO 10
30 PRINT "THE LENGTH OF THIS BLOCK IS ",RXLENGTH
!RUN <RETURN>
```
TPRINT

Description
The TPRINT command displays the contents of the trace buffer. Refer to Section 3.2 for a description of the Simulation buffers.

Type
Statement or Direct

Syntax
TPRINT Displays the trace buffer in hex.
TPRINT ASCII Displays the trace buffer in ASCII.
TPRINT EBCDIC Displays the trace buffer in EBCDIC.

Abbreviation
TP.

See Also
REC, LTPRINT

Example
This example builds and transmits a block of data and displays the trace buffer in hex, ASCII and EBCDIC.

10 $A= "THIS IS DATA" Assigns a string to $A.
20 TXBUFFER=$A Assigns $A to the transmission buffer.
30 TXSTATUS=128 Assigns a value to the transmission control status byte.
40 TRAN Transmits the block in the transmission buffer.
50 TPRINT Displays the trace buffer in hex.
60 TPRINT ASCII Displays the trace buffer in ASCII.
70 TPRINT EBCDIC Displays the trace buffer in EBCDIC.

!RUN <RETURN>
TRAN

Description  The TRAN command transmits a block from the transmission buffer (TXBUFFER), according to the framing defined by the transmission control status byte (TXSTATUS). Generally, TXBUFFER, TXSTATUS, and TRAN are used together in a program.

Type  Statement or Direct

Syntax  TRAN

Abbreviation  TR.

See Also  TXBUFFER, TXSTATUS

Example  This example builds and transmits a block of data and displays the trace buffer in hex.

```
10 $A = "THIS IS DATA"  Assigns a string to $A.
20 TXBUFFER = $A       Assigns $A to the transmission buffer.
30 TXSTATUS = 128      Assigns a value to the transmission control status byte.
40 TRAN                Transmits the block in the transmission buffer.
50 TPRINT              Displays the trace buffer in hex.
!RUN <RETURN>
```
**TXBUFFER**

**Description**
The TXBUFFER command defines the contents of the transmission buffer which assembles the data to be transmitted in a specified order. The data is stored in the transmission buffer prior to transmission and is transmitted according to the transmission control status byte (TXSTATUS).

The transmission buffer data can be a string, data that has been read from a disk-based data file, or other types of user-defined data.

**Type**
Statement or Direct

**Syntax**

| TXBUFFER = DLE          | Stores the indicated control character in the transmission buffer. |
| TXBUFFER = ACK          | Stores a string value in the transmission buffer. |
| TXBUFFER = 0            | Stores one or more hexadecimal values in the transmission buffer. |
| TXBUFFER = $A           | Stores a combination of control characters, strings and hex values. |
| TXBUFFER = &10, &70     | |

**Abbreviation**
TXB.

**See Also**
TXSTATUS, TRAN, TPRINT

**Example**
This example assigns a string to the transmission buffer, transmits it, and displays the trace in hex.

```
10 $A = "ABCDE"
20 TXBUFFER = $A
30 TXSTATUS = &BC
40 TRAN
50 TPRINT !RUN <RETURN>
```

(result) T 00:00:00:00:00 5 41 42 43 44 45
## TXSTATUS

**Description**  
TXSTATUS is a bit-mapped transmission control status byte that defines how the data in the transmission buffer will be transmitted. The contents of the transmission buffer is defined with the TXBUFFER command.

Each bit of the transmission control byte regulates a specific aspect of the protocol as illustrated in Figure 6.5-1 on the next page.

**NOTE:** Bit seven must always be set to one.

**Type**  
Statement or Direct

**Syntax**  
TXSTATUS = &xx

*where:* xx is the hexadecimal value for the transmission control byte. The ampersand (&) assigns the value in hex.

**Abbreviation**  
TXS.

**See Also**  
TXBUFFER
6.5: BSC-Commands

Bit 7  Bit 6  Bit 5  Bit 4  Bit 3  Bits 2 and 1  Bit 0
1      CRC   TEXT OR CONTROL  DLE   XPARENT OR NORMAL  EOT ETB or ETX  SOH or STX

START FRAMING CHARACTER
0 = SOH
1 = STX

END FRAMING CHARACTER
00 = EOT
01 = ETB
10 = ETX
11 = Illegal

TRANSPARENT TEXT ENABLE
1 = Enable transparent text mode
0 = Enable normal text mode

TRANSPARENT MODE 1
1 = Transparent mode 1 (DLE insertion)
0 = Transparent mode 0 (No DLE insertion)

TEXT MODE
1 = Text mode. Must be set for BCC accumulation on transmitted frame.
0 = 'Control' mode (No BCC)

CRC
1 = Send bad CRC
0 = Send good CRC

MUST BE 1

Figure 6.5-1: TXSTATUS Byte Bit Map
Example 1

```
TXSTATUS = &B5

A TXSTATUS value of B5 hexadecimal is represented in binary is:

1 0 1 1 0 1 0 1

Referring to Figure 6.5-1, this defines the transmission control byte as:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bits 2 and 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Good</td>
<td>CRC</td>
<td>TEXT</td>
<td>DLE</td>
<td>NORMAL</td>
<td>ETX</td>
</tr>
</tbody>
</table>
```

If the data in the transmission buffer is ABCDE, the data is transmitted as follows:

```
| SYN | SYN | DLE | STX | ABCDE | DLE | DLE | ETX | CRC1 | CRC2 |
```

Example 2

```
TXSTATUS = &AD

In binary this is represented by:

1 0 1 0 1 1 0 1

Referring to Figure 6.5-1, this defines the transmission control byte as:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bits 2 and 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Good</td>
<td>CRC</td>
<td>TEXT</td>
<td>NO</td>
<td>DLE</td>
<td>XParent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

If the data in the transmission buffer is ABCD DLE, the data is transmitted as follows:

```
| SYN | SYN | DLE | STX | ABCD | DLE | ETX | CRC1 | CRC2 |
```
Example 3

TXSTATUS = &A1

In binary this is represented by:

1 0 1 0 0 0 0 1

Referring to Figure 6.5-1, this defines the transmission control byte as:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bits 2 and 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Good CRC</td>
<td>TEXT</td>
<td>No DLE</td>
<td>NORMAL</td>
<td>EOT</td>
<td>SOH</td>
</tr>
</tbody>
</table>

If the data in the transmission buffer is ABCD DLE, the data is transmitted as follows (the DLE included is from the block of data in the transmission buffer):

| SYN | SYN | STX | ABCD DLE | ETX | CRC1 CRC2 |
Example 4

This program uses TXBUFFER to assign the data ABCDE in EBCDIC to the transmission buffer. It then transmits the data three different ways using different TXSTATUS bytes.

10 $A = "ABCD E" Stores ABCDE in string A.
20 $A = EBC$( $A ) Converts string A to EBCDIC.
25 TXBUFFER = $A Stores string A in the transmission buffer.
30 TXSTATUS = &A4 Defines the transmission control byte as A4 in hex.
40 TRAN Transmits the contents of the buffer using the TXSTATUS specification.
50 TXSTATUS = &80 Defines the transmission control byte as 80 in hex.
60 TRAN Transmits the contents of the buffer using the TXSTATUS specification.
70 TXSTATUS = &BC Defines the transmission control byte as BC in hex.
80 TRAN Transmits the contents of the buffer using the TXSTATUS specification.

(result) Line 40 TXSTATUS = &A4 = 1010,0100

SYN SYN SOH C1 C2 C3 C4 C5 ETX CRC PAD

Line 60 TXSTATUS = &80 = 1000,0000

SYN SYN C1 C2 C3 C4 C5 PAD

Line 80 TXSTATUS = &BC = 1011,1100

SYN SYN DLE SOH C1 C2 C3 C4 C5 DLE ETX CRC PAD
6.6 SAMPLE BSC PROGRAMS

The following BSC programs are for demonstration only. They show some major concepts to bear in mind when developing applications, but are not to be considered as tests for any particular application.

BSC1

This example polls a 3270, receives a poll. If there is no text to send, it replies with an EOT. Otherwise it sends the message, waits for an ACK, and closes transmission. Station addresses are: C1, C2, and C3.

```
10 DEFINE *POLL=*40
20 DEFINE *ADDR=*6C0
30 $F=CHRS(EOT)
40 GOSUB 290
50 $P=LEFT($P,1)
60 IF $P=$F GOTO 80
70 GOTO 40
80 GOSUB 290
90 IF LEN($P)>=5 GOTO 120
100 PRINT "NO POLLING RECEIVED-ABORTED"
110 STOP
120 $F=CHRS(POLL)+CHRS(ADDR)+CHRS(ADDR)+CHRS(ENQ)
130 $P=LEFT($P,5)
140 IF $P=$F GOTO 160
150 GOTO 100
155 REM
160 $A="THIS IS SOME EBCDIC DATA"
170 $A=EBCS($A)
180 TXSTATUS=&A2
190 TXBUFFER=$A
200 TRAN
210 GOSUB 290
215 REM
220 $P=LEFT($P,2)
230 $F=CHRS(DLE)+CHRS(ACK0)
240 IF $F=$P GOTO 270
250 PRINT "NO ACK RECEIVED"
260 GOTO 210
270 PRINT "END"
280 STOP
290 TIM2=20
300 REC $P, SY, $Z
310 IF RXLENGTH # 0 GOTO 350
320 IF TIM2#0 GOTO 300
330 PRINT "RX TIME OUT"
340 STOP
350 RETURN
```

10 Defines POLL mnemonic.
20 Defines ADDR mnemonic.
30 Execute subroutine at 290.
40 Execute subroutine at 290.
50 Poll received, send some data.
60 Print message, stop program.
70 Poll received, send some data.
80 Print message, stop program.
90 Poll received, send some data.
100 Print message, stop program.
110 Stop execution.
120 Defines POLL mnemonic.
130 Defines ADDR mnemonic.
140 Execute subroutine at 290.
150 Execute subroutine at 290.
160 Poll received, send some data.
170 Data transmitted, wait for an ACK.
180 Receive subroutine.
190 Data transmitted, wait for an ACK.
200 What have we received?
210 Terminate execution.
220 Reception subroutine.
230 What have we received?
240 If unable to receive, stop.
This program simulates a 3274 device. It receives polls and data, and sends the appropriate responses.

1 REM********AUTHOR ELIZABETH WALLING
5 L = 0
110 TIM2 = 60
120 REC $A, $B
130 IF TIM2 = 0 GOTO 900
140 IF RXLENGTH = 0 GOTO 120 Try again
150 IF RXLENGTH < 5 GOTO 440

160 REM Look for general poll.

170 SG = CHR$(&40) + CHR$(&40) + CHR$(&7F) + CHR$(&7F) + CHR$(&2D)
180 SD = LEFT$(SA,5)
190 IF SD = $G GOTO 500
205 REM Respond to general poll.

210 SG = CHR$(&60) + CHR$(&60) + CHR$(&40) + CHR$(&40) + CHR$(&2D)
220 IF SD = $G GOTO 400
230 REM Must be data.

235 SG = CHR$(&02)
240 SD = LEFT$(SA,1)
245 IF SD = $G GOTO 250
246 GOTO 900

250 IF RXLENGTH < 14 GOTO 550
255 SD = MID$(SA,10,7)
260 SG = "$SIGN ON"
265 SG = EB$(SG)
270 IF SD = $G GOTO 275
271 GOTO 310
275 REM Display last block and stop.

280 SD = MID$(SA,10,39)
285 SD = ASC$(SD)
290 PRINT SD
300 L = 1
305 GOTO 550
310 IF L = 1 GOTO 900
315 DISP

320 GOTO 550
400 REM Acknowledge the data

401 $S = CHR$(&10)+CHR$(&70)
405 TXBUFFER = $S
410 TXSTATUS = &80
415 TRAN

420 GOTO 110
440 REM Receive again.

441 REM Check for acknowledgement or end

of text. Is it an EOT?
7.1 INTRODUCTION TO CHAMELEON ASYNC

With Chameleon Async software, you can emulate and perform testing on an Asynchronous device. Programming commands specifically applicable to Async testing are a standard part of the software package and extend the Chameleon's BASIC command library.

All of the Chameleon BASIC commands are available in Async. In addition, Async includes special transmission, reception, timing control, and pin level interrogation and setting commands.

With Chameleon Async you can do the following:

• Emulate devices or networks to develop and test your software and systems in a working model of a live environment.

• Test a specific device or network by supplying either valid or invalid stimuli or responses.

• Perform positive testing by supplying your device or network with correct information to see if all procedures are working correctly.

• Perform negative testing by deliberately sending incorrect data in order to test error recovery and system/device susceptibility to failure induced by invalid data.

• Deviate from standard protocols during hardware/software development and compensate for protocol deviations in equipment supplied to you by others.
# 7.2 ASYNC PARAMETER SET-UP MENU

## Introduction

This section describes the parameters and valid values for the Async menu. This menu enables you to configure and save parameters for running Async simulation. For general information about using a parameter set-up menu, refer to Section 2.2.

For information about using the Save parameters menu, refer to Section 2.3.

---

## Async Parameter Set-Up Menu

**CHAMELEON ASYNC PARAMETER SET-UP**

- **A.** Simulate: DCE
- **B.** Parity: EVEN
- **C.** No. of Stop Bits: 1
- **D.** No. of Data Bits: 7
- **E.** Mode: Full Duplex

**F.** Baud Rate: 9600

**G.** Reception Mode: Block

**H.** End of Block Char: OD

---

**SELECT >**

**CHOICES:**

**INPUT:** SELECT by letter (A - H)

or type a control key shown below

**Z** = return to ASYNC, **?** = Help, **S** = Save parameters, **ESC** = Main Menu

---

The Async parameter fields and legal values are as follows:

**Simulate**

- DCE
- DTE

**Parity**

- EVEN
- NONE
- ODD

**No. of Stop Bits**

- 1
- 1.5
- 2
<table>
<thead>
<tr>
<th>No. of Data Bits</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Full Duplex</td>
<td>Half Duplex</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Do not use full duplex mode to transmit data from both sides of the line. This will cause problems since full duplex echoes back what it receives. Normal async usage is in half duplex mode.

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>50</th>
<th>1200</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>75</td>
<td>2400</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>4800</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>9600</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>19200</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reception Mode</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Character</td>
</tr>
</tbody>
</table>

End of Block Character: Enter a 2 digit hex number
7.3 ASYNC MNEMONIC TABLE

Introduction
This section describes the default mnemonic table for Chameleon Async. For a general description of the use of mnemonic tables, refer to Section 3.3.

Entries
The Async mnemonic table has four columns:

- Mnemonic name (1-6 characters)
- Decimal value
- Hexadecimal value
- Binary value

Table 7.3-1 on the next page shows the default Async mnemonic table. You can define and save your own mnemonic tables using the BASIC mnemonic commands described in Section 3.4.
<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACE</td>
<td>32</td>
<td>20</td>
<td>00100000</td>
</tr>
<tr>
<td>BELL</td>
<td>7</td>
<td>07</td>
<td>00000111</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>SOH</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>STX</td>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
<tr>
<td>ETX</td>
<td>3</td>
<td>03</td>
<td>00000011</td>
</tr>
<tr>
<td>EOT</td>
<td>4</td>
<td>04</td>
<td>00000100</td>
</tr>
<tr>
<td>ENQ</td>
<td>5</td>
<td>05</td>
<td>00000101</td>
</tr>
<tr>
<td>ACK</td>
<td>6</td>
<td>06</td>
<td>00000110</td>
</tr>
<tr>
<td>DLE</td>
<td>16</td>
<td>10</td>
<td>00010000</td>
</tr>
<tr>
<td>DC1</td>
<td>17</td>
<td>11</td>
<td>00010001</td>
</tr>
<tr>
<td>DC2</td>
<td>18</td>
<td>12</td>
<td>00010010</td>
</tr>
<tr>
<td>DC3</td>
<td>19</td>
<td>13</td>
<td>00010011</td>
</tr>
<tr>
<td>DC4</td>
<td>20</td>
<td>14</td>
<td>00010100</td>
</tr>
<tr>
<td>NAK</td>
<td>21</td>
<td>15</td>
<td>00010101</td>
</tr>
<tr>
<td>SYN</td>
<td>22</td>
<td>16</td>
<td>00010110</td>
</tr>
<tr>
<td>ETB</td>
<td>23</td>
<td>17</td>
<td>00010111</td>
</tr>
<tr>
<td>CAN</td>
<td>24</td>
<td>18</td>
<td>00011000</td>
</tr>
<tr>
<td>SUB</td>
<td>26</td>
<td>1A</td>
<td>00011010</td>
</tr>
<tr>
<td>ESC</td>
<td>27</td>
<td>1B</td>
<td>00011101</td>
</tr>
<tr>
<td>DEL</td>
<td>127</td>
<td>7F</td>
<td>01111111</td>
</tr>
<tr>
<td>BS</td>
<td>8</td>
<td>08</td>
<td>00001000</td>
</tr>
<tr>
<td>HT</td>
<td>9</td>
<td>09</td>
<td>00001001</td>
</tr>
<tr>
<td>LF</td>
<td>10</td>
<td>0A</td>
<td>00001010</td>
</tr>
<tr>
<td>VT</td>
<td>11</td>
<td>0B</td>
<td>00001011</td>
</tr>
<tr>
<td>FF</td>
<td>12</td>
<td>0C</td>
<td>00001100</td>
</tr>
<tr>
<td>CR</td>
<td>13</td>
<td>0D</td>
<td>00001101</td>
</tr>
<tr>
<td>SO</td>
<td>14</td>
<td>0E</td>
<td>00001110</td>
</tr>
<tr>
<td>SI</td>
<td>15</td>
<td>0F</td>
<td>00001111</td>
</tr>
<tr>
<td>EM</td>
<td>25</td>
<td>19</td>
<td>00011001</td>
</tr>
<tr>
<td>FS</td>
<td>28</td>
<td>1C</td>
<td>00011100</td>
</tr>
<tr>
<td>GS</td>
<td>29</td>
<td>1D</td>
<td>00011101</td>
</tr>
<tr>
<td>RS</td>
<td>30</td>
<td>1E</td>
<td>00011110</td>
</tr>
<tr>
<td>US</td>
<td>31</td>
<td>1F</td>
<td>00011111</td>
</tr>
</tbody>
</table>

Table 7.3-1: Async Mnemonic Table
7.4 COMMAND INDEX

Introduction

This section lists the Chameleon Async commands by function. If you need a command that performs a specific function, but you do not know the name of the command, you should be able to locate the command by referring to one of these tables.

The tables contain a brief description of the command and the command type (Statement or Direct). For more information about a command, refer to the page number indicated.

TABLE 7.4-1: ASYNC READ-ONLY VARIABLES

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAMING</td>
<td>7.5-5</td>
<td>Variable</td>
<td>Indicates if stop bits encountered.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>7.5-6</td>
<td>Variable</td>
<td>Indicates length of received block.</td>
</tr>
<tr>
<td>PARITY</td>
<td>7.5-8</td>
<td>Variable</td>
<td>Detects parity errors.</td>
</tr>
<tr>
<td>RXBREAK</td>
<td>7.5-10</td>
<td>Variable</td>
<td>Indicates if break sequence encountered.</td>
</tr>
</tbody>
</table>

TABLE 7.4-2: ASYNC TRANSMISSION AND RECEPTION COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAK</td>
<td>7.5-2</td>
<td>S or D</td>
<td>Transmits a break sequence.</td>
</tr>
<tr>
<td>CRC16</td>
<td>7.5-3</td>
<td>S or D</td>
<td>Calculates the 2-byte CRC for a string.</td>
</tr>
<tr>
<td>FOXMESS</td>
<td>7.5-4</td>
<td>D</td>
<td>Transmits the standard FOX message.</td>
</tr>
<tr>
<td>LRC</td>
<td>7.5-7</td>
<td>S or D</td>
<td>Calculates a 1-byte LRC for a string.</td>
</tr>
<tr>
<td>REC</td>
<td>7.5-9</td>
<td>S or D</td>
<td>Receives a block or character.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>7.5-11</td>
<td>S or D</td>
<td>Displays the contents on the trace buffer.</td>
</tr>
<tr>
<td>TRAN</td>
<td>7.5-12</td>
<td>S or D</td>
<td>Transmits string and characters</td>
</tr>
</tbody>
</table>
7.5 ASYNC COMMANDS

Introduction This section provides a complete description of the Async commands. The commands are organized alphabetically, one command per page.

If you know the name of the command you want to use, you can look up the command in this section. If you are unsure of the command you need, refer to the previous section which lists the commands by function.

The following information is provided in this section:

- Command description
- Type of command (Statement or Direct)
- Syntax
- Command abbreviation
- Related commands (See Also)
- Example(s) of command usage
### BREAK

**Description**  
The BREAK command transmits a BREAK sequence.

**Type**  
Statement or Direct

**Syntax**  
BREAK

**Abbreviation**  
B.

**See Also**  
RXBREAK

**Example**  
BREAK
## CRC16

### Description
The CRC16 function calculates the CRC (Cyclic Redundancy Check) characters for a string of data. The polynomial used is:

\[ x^{16} + x^{15} + x^2 + 1 \]

### Type
Statement or Direct

### Syntax
\[ \$B = \text{CRC16}($A) \]

*where*: $A$ is the string of data and $B$ returns two bytes which are the CRC characters for the data in $A$.

### Abbreviation
None

### See Also
LRC

### Example
```
10 $A = "HELLO"
20 $B = \text{CRC16}($A)
30 \text{PRINT} $B
```

(result) Aa
FOXMESS

Description
This command transmits the standard FOX message and repeats it until the operator hits any key.

The fox message that is transmitted is:

(STX)THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS BACK 0123456789 (CR)(LF)(ETX)

To allow the greatest possible density and the fastest possible transmission, data transmitted does not appear in the trace buffer and therefore cannot be displayed using the TPRINT command.

Type
Direct

Syntax
FOXMESS

Abbreviation
F.

Example
FOXMESS
(result) HIT ANY KEY TO STOP FOX MESSAGE
FRAMING

Description  FRAMING is a numeric read-only variable that detects the presence of stop bits at the end of the received block. FRAMING has the following valid values:

- FRAMING = 0 One or more valid stop bits detected.
- FRAMING = 1 No stop bits detected.

Type Variable

Syntax (See examples below.)

Abbreviation FRA.

See Also REC

Example 1 PRINT FRAMING

Example 2 IF FRAMING = 1 PRINT "NO STOP BITS"
LENGTH

Description  LENGTH is a numeric read-only that returns the number of characters received in a block. If LENGTH = 0, it indicates that nothing has been received.

Type  Statement or Direct

Syntax  (See examples below.)

Abbreviation  L.

See Also  REC

Example 1  PRINT LENGTH

Example 2  10 REC
            20 IF LENGTH = 0 GOTO 10
LRC

Description  LRC is a function that calculates the LRC (Longitudinal Redundancy Check) for a string of data. The LRC computation command checks the parity selection chosen in the Async Parameter Set-Up menu. It uses the polynomial:

$$x^8 + 1$$

Type  Statement or Direct

Syntax  $$X = \text{LRC}(Y)$$

where: $Y$ is the string of data and $X$ returns a single character that equals the LRC for the data in $Y$.

Abbreviation  None

See Also  CRC16

Example 1  This example prints the hex value of the LRC of the string HELLO.

10 $A = "HELLO"
20 $B = \text{LRC}($A$)
30 PRINT %$B

(result)  42

Example 2  This example prints the hex value of the LRC of the string GOODBYE.

10 $A = "GOODBYE"
20 $B = \text{LRC}($A$)
30 PRINT %$B

(result)  DD
PARITY

Description
PARITY is a numeric read-only variable that indicates whether a parity error has occurred. PARITY has the following valid values:

- PARITY = 0 No parity error detected.
- PARITY = 1 parity error detected.

Type
Variable

Syntax
(See examples below.)

Abbreviation
P.

See Also
REC

Example 1
PRINT PARITY

Example 2
IF PARITY = 0 PRINT "NO PARITY ERROR DETECTED"
**REC**

**Description**
REC assigns the next character (if in character mode) or block of characters (if in block mode) to the string variables as specified in the command. A string variable can be assigned a maximum of 255 characters. When the 255-character maximum is reached, data is stored in the next string variable indicated.

After a REC command, these Async numeric read-only variables are updated: LENGTH, PARITY, FRAMING, RXBREAK.

**Type**
Statement or Direct

**Syntax**
REC $A, $B,...

*where: $A, $B, are the string variables that the received data is assigned to.*

**Abbreviation**
None

**See Also**
FRAMING, LENGTH, PARITY, RXBREAK

**Example**
10 REC $A, $B
20 IF LENGTH = 0 GOTO 10
RXBREAK

Description  RXBREAK is a numeric read-only variable that indicates if a break sequence has been received. It has the following valid values:
  • RXBREAK = 0 No break sequence received.
  • RXBREAK = 1 Break sequence received.

Type  Variable

Syntax  (See examples below.)

Abbreviation  RX.

See Also  REC

Example 1  PRINT RXBREAK

Example 2  IF RXBREAK = 1 PRINT "BREAK SEQUENCE RECEIVED"
TPRINT

Description
TPRINT prints the contents of the trace buffer to the screen.
The trace buffer contains the data received or transmitted in ASCII.

Type
Statement or Direct

Syntax
TPRINT
Prints the trace in ASCII.
TPRINT HEX
Prints the trace in hexadecimal.

Abbreviation
TP.

See Also
REC, TSAVE, TLOAD, TPRINT, LTPRINT

Example 1
This example displays the contents of the trace buffer in ASCII.
TPRINT
(result)
T 15:09:38:31:60 61
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS BACK 0123456789
CR

Example 2
This example displays the contents of the trace buffer in hex.
TPRINT HEX
(result)
T 15:09:38:31:60 61
54 48 45 20 51 55 49 43 48 20 42 52 4F 57 4E 20 46 4F 58 20
4A 55 40 50 53 20 4F 56 45 52 20 54 48 45 20 4C 41 5A 59 20
44 4F 47 53 20 42 41 43 4B 20 30 31 32 33 34 35 36 37 38 39
0D
# TRAN

**Description**  
TRAN transmits string data, mnemonics or literal data.

**Type**  
Statement or Direct

**Syntax**  
- `TRAN $A...`  
  Transmits $A.
- `TRAN CR, LF...`  
  Transmits the mnemonics CR (carriage return) and LF (line feed).
- `TRAN "ABCD"...`  
  Transmits ABCD.
- `TRAN $A, CR, LF, "ABCD"`  
  Transmits a combination of the above.

**Abbreviation**  
TR.

**See Also**  
REC

**Example**  
10 $A = "HELLO"
20 $B = "GOODBYE"
30 TRAN $A, ",", $B, CR, LF
7.6 ASYNC SAMPLE PROGRAMS

Async 1

The program transmits the fox message that is being rotated as data.

```
10 $A="THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS BACK 0123456789"
20 GOTO 90
30 $B=RIGHTS($A,(LEN($A)-1))
40 $B=$B+LEFTS($A,1)
50 $A =$B
60 RETURN
70 FOR Y = 1 TO 23
80 FOR X = 0 TO LEN($A)
90 TRAN $A,CR
100 GOSUB 30
110 NEXT X
120 TRAN LF
130 NEXT Y
```

Transmission subroutine.
Deletes first character of $A and stores the rest in $B.
Adds first character to end of $B.
Stores new string in $A.
End of rotation, returns to line 115.
Complete rotation is done 23 times.
One rotation depends on length of $A.
Transmits current string and carriage return.
Goes to rotation subroutine.
Increments X.
Transmits line feed.
Increments Y.
Async2

This program transmits a fox message that is being rotated as data and then writes the data to a file.

```
10 OPEN "O","B:DATA"
20 TRAN SUB
30 $A="THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS BACK0123456789"
40 GOTO 90
50 $B=RIGHTS($A,(LEN($A)-1))
60 $B=$B+LEFTS($A,1)
70 $A = $B
80 RETURN
90 FOR Y = 1 TO 23
100 FOR X = 0 TO LEN($A)
110 TRAN $A, CR
120 WRITE $A
130 GOSUB 30
140 NEXT X
150 TRAN LF
160 NEXT Y
```

Transmits and writes sub-routine.

Deletes the first character of $A and stores the rest in $B.

Adds the first character to end of $B.

Places new string in A.

End of rotation; returns to line 115.

Complete rotation is done 23 times.

One rotation depends on the length of string A.

Transmits current string and carriage return.

Writes $A to output file, DATA.

Rotation sub-routine.

Increments X.

Transmits line feed.

Increments Y.
Async3

This program demonstrates the use of the Async mnemonic table. It shows the results of checking the mnemonic table prior to transmitting a one-character string converted to its ASCII equivalent.

5 \$A = " "

Initializes string A to zero characters.

10 FOR \$X = 0 TO \&20

Sets up a loop, incrementing \$X from 0 to 20 Hex.

20 \$A = \$A + CHR$(\$X)

CHR$ returns the ASCII value of the loop index \$X and stores it in string A.

30 NEXT \$X

Increments loop index.

40 TRAN \$A

Transmits string A, the ASCII value of \$X, according to the mnemonic table.

50 TPRINT

Displays the string as stored in the trace after transmission.

60 TPRINT HEX

Displays the trace in hexadecimal.

The results of the sample program are:

```
T 13:46:58:41:30 33
  NULL 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
  14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
```

In the first display, the results of TPRINT are shown as the standard Async protocol mnemonics from the table. The second display shows the transmitted mnemonics displayed in hexadecimal. T 13:46:58:41:30 33 indicates the time of the transmission and that the transmission block is 33 bytes long.
Async4

The example program below illustrates how these variables can be tested following reception. The program uses the shift left operation to calculate the line number for the GOTO statement based on the values in the variables.

```
10 REC SA
20 IF LENGTH = 0 GOTO 10

30 GOTO 1000 + (10 * ((PARITY OR (FRAMING SHL 1)) OR (RXBREAK SHL 2)))

1000 PRINT "RECEIVED O.K."

1005 GOTO 10
1010 PRINT "PARITY ERROR"

1015 GOTO 10
1020 PRINT "FRAMING ERROR"

1025 GOTO 10
1030 PRINT "PARITY & FRAMING ERROR"

1035 GOTO 10
1040 PRINT "BREAK DETECTED"

1045 GOTO 10
1050 PRINT "BREAK & PARITY ERRORS"

1055 GOTO 10
1060 PRINT "BREAK & FRAMING ERRORS"

1065 GOTO 10
1070 PRINT "BREAK, FRAMING & PARITY ERRORS"

1075 GOTO 10
```

Receive from the line. If nothing was received, try again.

All variables = 0; no errors detected. Print line number 1000.

Parity error detected. Print line number 1010.

Framing error detected. Print line number 1020.

Parity and Framing errors. Print line number 1030.

Break sequence detected. Print line number 1040.

Break and parity errors. Print line number 1050.

Break and framing errors. Print line number 1060.

Break, framing and parity errors. Print line number 1070.
8.1 INTRODUCTION TO SITREX

Introduction
SITREX is the Chameleon X.25 Simulator. It can simulate an X.25 Network or Subscriber and provide a live, certified environment for the development and testing of X.25 products.

This software package can handle all X.25 procedures automatically, including:

- Flow control at the frame and packet levels
- Error detection and error recovery
- Data generation and reception
- All protocol timers and counters

X.25 is a very specific, well-defined protocol. The Chameleon's X.25 Simulator complies entirely with the CCITT specifications and allows you to control levels one (Physical interface), two (Link level), and three (Packet level) independently.

Note
This chapter assumes that you have a thorough understanding of the X.25 protocol.

Standards
SITREX supports the CCITT 1984 X.25 standard, including the three network-specific implementations: NTT, DATAPAC, and TRANSPAC.

TRANSPAC SITREX is identical to CCITT SITREX, except you can transmit and receive calls on LCN 0.
SITREX ENVIRONMENT

Sitrex has a multi-tasking operating system that provides the following features and capabilities:

- Automatic X.25 Simulator
- Trace Buffer Function
- Command Language

You can also use the three set-up menus to configure physical level parameters, frame/packet level parameters, and user facilities for SITREX simulation. Refer to Section 8.3 for a description of these menus.

This section describes how the three major SITREX features interact to provide you with a comprehensive X.25 simulation environment. Each of the features is described briefly, as an overview. The sections that contain detailed information about each feature are referenced.
Automatic X.25 Simulator

The Automatic X.25 Simulator responds automatically to received traffic at packet and/or frame level, or not at all, according to CCITT specifications.

The Automatic X.25 Simulator can be used without any programming to create and monitor normal data communications traffic on the line. It produces valid X.25 responses to all incoming data, and updates counters and timers.

There are three possible idle states for the Automatic X.25 Simulator, as configured in the Mode parameter in the Sitrex Physical Level Parameter Set-Up menu:

- Initiative - The Simulator attempts to initiate an X.25 link, by sending a DISc or DM on the idle line.
- Standby - The Simulator reacts to incoming traffic.
- Manual - The Simulator is inactive; frame level is off.

Section 8.2 describes actions of the Automatic X.25 Simulator in detail.
Trace Buffer

The Trace Buffer stores and displays the transmitted and received traffic. You can specify that the traffic be interpreted at frame level, frame and packet level, or not at all.

You can turn the trace buffer on and off from the Automatic X.25 Simulator or by using the command language. Traffic is stored in the trace buffer only if the trace is on and there is space available in the buffer.

The information in the trace buffer can be saved to disk, displayed on the screen, or output to a printer. Refer to Section 8.4 for more information about the trace buffer.
The SITREX command language provides you with additional control over the X.25 simulation environment. The command language can be used in conjunction with the Automatic X.25 Simulator to override the Simulator's automatic functions.

For example, frame and packet level processing can be turned ON or OFF independently. This ability to manipulate the Simulator enables you to check error response and recovery procedures in an X.25 environment.

You can generate scenarios that work with or without the Automatic X.25 Simulator to create normal or abnormal situations to troubleshoot an X.25 Subscriber or Network, or deviate from the X.25 standards.

Sections 8.5 - 8.7 describe the use of the Sitrex command language.

Figure 8.1-1 illustrates the major features of the Sitrex X.25 simulation environment. It illustrates how to access each of the Sitrex features, and how to turn them on and off. The text that follows the illustration describes how the features interact with each other to process incoming traffic.
Figure 8.1-1: Sitrex Simulation Environment
A. Applications Selection Menu

In the Applications Selection Simulation window, load SITREX or T\_SITREX (Transpac Sitrex) on the desired port. Press Go to start the simulator and access the Parameter Set-Up Menus.

B. Parameter Set-Up Menus

These menus enable you to configure Physical (level 1) parameters, Frame and Packet (level 2 and 3) parameters and User Facilities. See Section 2.2 and 2.3 for general instructions for the set-up menus. See Section 8.3 for a description of the Sitrex set-up menus.

These parameter settings affect the Automatic X.25 Simulator. They also affect scenarios you write in the Sitrex command language, unless a parameter command overrides the menu setting. Refer to the parameter commands in Section 8.7 for more information.

From any of the set-up menus, press Z to access Simulation or press Esc to return to the Main Menu.

C. Chameleon Simulator Active

This message indicates that you can begin your simulation activities. The Mode option in the Physical Parameters Set-up menu (B) determines whether the Automatic X.25 Simulator is active. Mode options are:

- Initiative - Automatic X.25 Simulator attempts to set up an X.25 link.
- Standby - Automatic X.25 Simulator reacts to incoming frames.
- Manual - Automatic X.25 Simulator is off. There is no automatic reaction to incoming frames, except through the use of the command language.

When active (Initiative or Standby mode), you can impose control over the reactions of the Automatic X.25 Simulator using the command language. See F - H, below for more information.

There are a limited number of commands that can be entered at this prompt. These commands enable you to turn the trace mode and command language mode on and off. See Section 8.4 for a list of these commands.

D. Trace Buffer

The Trace buffer can be turned on and off from the Simulator. To turn the trace on, type PP and press RETURN. (Note that the first character you type appears as % on the screen.) This causes the traffic received and transmitted on the line to be stored and displayed in an interpreted format in the trace buffer.
Traffic is stored in the trace buffer only if the trace is active and there is adequate space available in the buffer. To make the trace buffer idle (stop displaying traffic) type CTRL P.

You can also turn the trace buffer on and off using the command language. Refer to the STON (set trace on) and STOF (set trace off) commands in Section 8.7 for more information.

Refer to Section 8.4 for a more detailed description of the trace buffer and interpreting the display.

**E. Command Language**

To access the command language from the SITREX simulator, type PS and press RETURN. (Note that the first character you type appears as % on the screen.) This displays the ! prompt, at which you can enter commands and write scenarios. Refer to Sections 8.5 - 8.7 for a description of the command language.

A scenario reacts to incoming traffic only if it is WAITing for a frame or packet. Refer to the WAIT commands in Section 8.7 for more information.

You can use the command language to control the Automatic X.25 Simulator. Specifically, you use commands that control whether the Automatic X.25 Simulator processes frame level only, frame level and packet level, or neither level. Refer to Section 8.3 for a description of the commands that affect the Automatic X.25 Simulator.

To exit the command language mode, but remain in the Simulator, use the EXIT command. To leave SITREX and return the main menu, use the HALT command.

**F. Line Interface**

**Level 1**

Level 1 acquires and transmits data to and from the line. The Physical Parameter Set-Up menu controls the parameters relevant to this level. For example, you can determine the baud rate and control whether the Chameleon is a DCE or DTE. The settings in the Physical Level Parameter Set-Up menu (B) affect this level of simulation.

You can use the command language to send various types of frames. For example, Figure 8.1-1 shows the FB command which transmits a user-defined frame in binary. Refer to the Frame Level Commands in Section 8.7 for more information.
G. Level 2
HDLC

HDLC is frame level. The following actions can occur:

The Automatic X.25 Simulator processes the frame, sends an appropriate response to the frame if the mode is Initiative or Standby, and the frame level is on (SFON command). If frame level is off (SFOF command), the frame is ignored.

Scenario. If a scenario has a WAIT (frame) command, the received traffic is compared to the frame specified in the WAIT command. This determines the next action in the scenario. If there is no WAIT command, the scenario does not react to incoming traffic.

The SFOF and SFON commands determine whether the Automatic X.25 Simulator processes incoming frames.

You can also control the link using the SLON (set link on) and SLOF (set link off) commands.

H. Level 3
X.25

X.25 is packet level. The following actions occur:

Automatic X.25 Simulator processes the packet, sends an appropriate response to the frame if Mode is Initiative or Standby, and the packet level is on (SPON command). If packet level is off (SPOF command), the packet is ignored.

Scenario. If a scenario has a WAIT (packet) command, the received traffic is compared to the packet specified in the WAIT command. This determines the next action in the scenario. If there is no WAIT command, the scenario does not react to incoming traffic.

The SPOF and SPON commands determine whether the Automatic X.25 Simulator processes incoming packets.

You can use the command language to send various types of packets. For example, Figure 8.1-1 shows the PRST command which transmits a RESTART packet. Refer to the Packet Level Commands in Section 8.7 for more information.

I. Pseudo-Users

Pseudo-Users. SITREX supports up to seven pseudo-users. Each pseudo user simulates a network device connected to a logical channel. You can use the command language to place calls from a pseudo-user to any network address. For example, Figure 8.1-1 places a call from pseudo-user number 5.
OTHER SITREX FEATURES

Memory Management

SITREX uses a dynamic memory allocation scheme to maximize the efficiency of the Chameleon. These SITREX features affect the amount of memory available to you:

- Trace buffer
- Active Pseudo-Users
- Scenario length
- Buffers (frame, retransmission, etc.)

You can control the amount of memory you are using by managing the length of your scenarios.

You can also control the size of the trace buffer using the trace buffer commands in Section 8.7. Specifically, these commands can be used to control the size of the trace buffer:

- Clear the trace (TRACE command)
- Turn the trace off/on (STOF/STON commands)
- Limit the length of the data (STR command)

Timers and Counters

SITREX allows you to set timers and counters using either the parameter set-up menu or command language. For example, both the T1 timer and the N2 counter can be set from the parameter set-up menu or using a set command.

State Variables

There are several variables in X.25, such as timeouts and frame window size, that control the flow of traffic. The State variable refers to the current value of these variables. State variables are controlled by the Automatic X.25 Simulator.

SITREX continually updates the state variables and the Automatic X.25 Simulator takes corrective action depending on their values.

For example, the high-level command, SLON, transmits a SABM and waits for a response. If a valid response is not received within the T1 time limit, the Automatic X.25 Simulator transmits a polled SABM and decrements N2. This continues until N2 is 0, at which time the Automatic X.25 Simulator attempts to set the link off.
SITREX Circuit Management

You can choose the number of Switched Virtual Circuits (SVC) and Permanent Virtual Circuits (PVC) for your application using the parameter set-up menu. You can assign addresses to SVCs in command mode. You can also assign a logical channel for both PVCs and SVCs.

SITREX will assign a channel for SVCs only. The Automatic X.25 Simulator assigns a Logical Channel Number (LCN) based on whether the Chameleon is configured as a Network or a Subscriber, according to the CCITT specifications.

To enable you to reassign a PVC, a PVC is removed by sending or receiving a CLEAR.

Note

It is possible for PVC's and SVC's to overlap. However, this is not recommended. Also, SITREX will automatically handle only seven PVCs or SVCs at one time (out of a maximum of 16 groups of 255).
8.2 SITREX AUTOMATIC X.25 SIMULATOR

Introduction

This section describes the Automatic X.25 Simulator. The Automatic X.25 Simulator can be used without any programming to create and monitor normal data communications traffic on the line. It produces valid X.25 responses to all incoming data, and updates counters and timers.

Accessing SITREX

To access the Automatic X.25 Simulator, follow these steps:

1. From the Chameleon main menu, use the softkeys to select SITREX. (To use the TRANSPAC implementation of SITREX, select T.Sitrex from the menu.)

2. Press F2 (Menu) to access the set-up menus.

3. In the Physical Parameter Set-Up menu, select the Mode setting that corresponds to the desired Automatic X.25 Simulator mode. The options are:

   • Initiative - Automatic X.25 Simulator attempts to set up an X.25 link.
   • Standby - Automatic X.25 Simulator reacts to incoming frames.
   • Manual - Automatic X.25 Simulator is off. There is no automatic reaction to incoming frames, except through the use of the command language.

4. If you want to select the NTT or DATAPAC implementations of SITREX, change the value of the Sitrex Version parameter in the Frame & Packet Level Parameter Set-Up menu.

5. Change any other parameters as required and press Z to load the simulator. (Refer to Section 8.3 for a description of the menus.)
6. The message SITREX SIMULATOR ACTIVE appears and the blinking cursor is displayed. The Automatic X.25 Simulator is now active according to the mode you selected in the Physical Set-Up menu.

From here, you can activate the trace buffer, enter command language mode, or exit Simulation using the commands in Table 8.2-1. (Note that the first character you type appears as a % on the screen.)

Refer to Section 8.4 for a description of the trace buffer. Refer to Sections 8.5 - 8.7 for a description of the SITREX command language.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Sets Simulator to echo Called and Calling Addresses in Call Confirmation packets.</td>
</tr>
<tr>
<td>PP</td>
<td>Activates Trace Buffer so that traffic is stored and displayed. (Once the trace is active, use CTRL P to make the trace idle.)</td>
</tr>
<tr>
<td>PS</td>
<td>Enters command language mode and displays ! prompt enabling you to enter Sitrex commands and write scenarios. (From the ! prompt, you can exit SITREX using the HALT command or exit command language mode using EXIT.)</td>
</tr>
</tbody>
</table>

Table 8.2-1: Automatic X.25 Simulator Options
Pseudo-Users

A pseudo-user is a logical subscriber supported by SITREX. SITREX can support up to seven pseudo-users. Each pseudo-user simulates a physical device connected to a logical channel.

Two of these are internally reserved and pre-assigned by SITREX for the local terminals and cannot be programmed. Their default conditions are the following:

01 Associated with the Chameleon Trace page (pink window)
02 Associated with the Chameleon Simulation page (blue window)

You can program pseudo-users 03 - 07 to re-configure them as one of the following:

- Traffic generators (Sends a continuous stream of ASCII data packets)
- Data absorbers (Normal terminals)
- Echo generators (Retransmits all the user data received on the same logical channel)

The default configuration for these pseudo-users is:

03 Traffic generator
04 Echo generator
05 Data absorber
06 Second traffic generator
07 Third traffic generator

Pseudo-User Addressing

According to the CCITT X.25 specifications, one byte of address information providing specific local terminal addressing can be added to a transmitted address in a CALL packet. This byte is used to address a specific pseudo-user.

For example, if the PAD address is 12345, and one terminal (pseudo-user) is 07, the terminal is addressed as 12345 07.
CONTROLLING X.25 LEVELS

You can control whether the X.25 frame and packet levels are processed by the Automatic X.25 Simulator using the following SITREX commands:

- SET FRAME LEVEL ON (SFON)
- SET FRAME LEVEL OFF (SFOF)
- SET PACKET LEVEL ON (SPON)
- SET PACKET LEVEL OFF (SPOF)

SET FRAME LEVEL ON (SFON)

When frame level is ON, the Automatic X.25 Simulator processes incoming frames and updates frame level state variables accordingly. The Automatic X.25 Simulator can send frames in response to incoming frames or when a timer has expired. You can also send frames manually using SITREX commands in command mode. Frame level must be on if packet level is on.

SET FRAME LEVEL OFF (SFOF)

When frame level is OFF, the Automatic X.25 Simulator does not process incoming frames, and the frame level state variables do not get updated with incoming traffic. You can process incoming frames using SITREX commands in command mode.

SET PACKET LEVEL ON (SPON)

When packet level is ON, the Automatic X.25 Simulator processing incoming packets and updates packet level state variables accordingly.

SET PACKET LEVEL OFF (SPOF)

When packet level is OFF, the Automatic X.25 Simulator does not process incoming packets, but you can process them using SITREX commands in command mode.

Setting the frame or the packet level OFF does not make the Automatic X.25 Simulator inactive. It affects it so that the reception of frames or packets is disabled. SITREX's automatic actions will continue to be occur.
The example below shows the result of a TPRINT (trace print) command when the frame level is off. It indicates the frames that were transmitted and received, but shows that any frames received were ignored by the Automatic X.25 Simulator.

TPRINT
T-002 03 SABM \leftrightarrow SLON
R-002 03 UA \leftrightarrow SFOF
T-002 03 PSABM \leftrightarrow Received a UA, but with frame level off, the frame is ignored.

Since Simulator never sees the UA, it again attempts to establish the link.

Applications
Using the SFOF/SFON and SPOF/SPON commands, there are three possible cases that can occur during SITREX simulation:

- Frame level OFF/Packet level OFF
- Frame level ON/Packet level OFF
- Frame level ON/Packet level ON

Note that Frame level OFF/Packet level ON is not valid, since packets cannot be processed if frames are ignored.

Each of these cases is described on the following pages.
Case 1: FRAME LEVEL OFF (SFOF) / PACKET LEVEL OFF (SPOF)

With both the frame and packet level OFF, the Automatic X.25 Simulator is disabled. None of the State variables are updated, and none of the incoming frames or packets are processed by the Automatic X.25 Simulator. However, incoming frames and packets can be processed by a SITREX scenario.

For example, if you establish the link with the SLON command, and then turn OFF the frame level with the SFOF command, the Automatic X.25 Simulator thinks that the link is established, but does not process incoming frames.

If you transmit I-Frames (FIPH command), the Automatic X.25 Simulator inserts the correct N(s) for each frame sent.

When the DUT sends an RR, the Automatic X.25 Simulator does not process it because the frame level is off.

<table>
<thead>
<tr>
<th>SITREX COMMAND</th>
<th>SIMULATOR SENDS</th>
<th>DUT SENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLON</td>
<td>SABM</td>
<td></td>
</tr>
<tr>
<td>SFOF</td>
<td></td>
<td>UA</td>
</tr>
<tr>
<td>FIPH</td>
<td>I-FRAMES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RR s</td>
</tr>
</tbody>
</table>

(Not processed by SITREX)
Case 2: FRAME LEVEL ON (SFON) / PACKET LEVEL OFF (SPOF)

With the frame level ON and the packet level OFF, incoming frames are processed by the automatic Simulator at the frame level only. For example, frame level State variables ($N(s)$ and $N(r)$) are updated.

The packet portion of an incoming frame is not processed by the Automatic X.25 Simulator and packet level state variables, such as $P(r)$, are not updated. However, packet level State variables relevant to outgoing packets are updated.

When transmitting packets with the packet level OFF, only low-level commands, such as PHh1h2 can be used. In the example below, SITREX sends a RESTART packet and the DUT sends a RESTART CONFIRMATION packet in return. If the packet level is OFF, SITREX does not process the RESTART CONFIRMATION. After the timeout period, SITREX sends another RESTART.

Incoming packets can be processed using SITREX commands in command mode. So in this example, you could override the Automatic X.25 Simulator using a command that causes SITREX to wait for and process a RESTART CONFIRMATION packet. (WPCRST)

<table>
<thead>
<tr>
<th>SITREX COMMAND</th>
<th>SIMULATOR SENDS</th>
<th>DUT SENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRST</td>
<td>RESTART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>← RESTART CONFIRMATION</td>
<td></td>
</tr>
<tr>
<td>PRST</td>
<td>RESTART</td>
<td></td>
</tr>
<tr>
<td>SPOF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WPCRST</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>← RESTART CONFIRMATION</td>
<td></td>
</tr>
</tbody>
</table>

(180 seconds)

RESTART
Case 3: FRAME LEVEL ON (SFON) / PACKET LEVEL ON (SPON)

With both the frame and packet level ON, all incoming frames and packets are processed, and all of the State variables are updated.

In the example below, SITREX sends a RESTART and the DUT responds with a RESTART CONFIRMATION.

Since both packet level and frame level are on, SITREX processes the RESTART CONFIRMATION and sends a CALL. The DUT responds with a CALL CONFIRMATION. SITREX continues the communication by sending an RR.

<table>
<thead>
<tr>
<th>SITREX COMMAND</th>
<th>SIMULATOR SENDS</th>
<th>DUT SENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRST</td>
<td>RESTART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>← RESTART CONFIRMATION</td>
<td></td>
</tr>
<tr>
<td>PU1CALL2</td>
<td>CALL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>← CALL CONFIRMATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td></td>
</tr>
</tbody>
</table>
Set Link ON/OFF Commands

The Set Link ON/OFF commands (SLON/SLOF) affect the internal state of SITREX, while the Frame commands send specified frames, without updating the internal State variables of SITREX. This leads to the following situation:

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>Chameleon</th>
<th>USER'S EQUIPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLON</td>
<td>SABM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>← UA</td>
</tr>
<tr>
<td></td>
<td><strong>Chameleon IN INFORMATION TRANSFER STATE</strong></td>
<td></td>
</tr>
<tr>
<td>FDISC</td>
<td>DISC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>← UA</td>
</tr>
<tr>
<td>FRMR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the above case, the FDISC command entered through SITREX will be transmitted, but the internal State variables will not be updated. Consequently, when the user's equipment transmits a UA, the frame will be rejected, because the automatic Simulator sees it as unsolicited UA.

To allow SITREX to enter the down state, the SLOF command should have been used as shown below. This is true for all user-defined frames.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>Chameleon</th>
<th>USER'S EQUIPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLON</td>
<td>SABM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>← UA</td>
</tr>
<tr>
<td></td>
<td><strong>Chameleon IN INFORMATION TRANSFER STATE</strong></td>
<td></td>
</tr>
<tr>
<td>SLOF</td>
<td>DISC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>← UA</td>
</tr>
<tr>
<td></td>
<td><strong>Chameleon IN DOWN STATE</strong></td>
<td></td>
</tr>
</tbody>
</table>
Consider the following case (assuming SLON was used):

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>Chameleon</th>
<th>USER'S EQUIPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PH10011300</td>
<td>CLEAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLEAR CONF.</td>
</tr>
<tr>
<td>CLEAR</td>
<td></td>
<td>CLEAR CONF.</td>
</tr>
</tbody>
</table>

At first, this may appear to be incorrect because the Chameleon should not have transmitted the second CLEAR. What actually happened is that the user-defined frame is transmitted without updating the Chameleon’s State variables. Consequently, the received CLEAR CONFIRMATION is cleared by the automatic Simulator.
Set Packet Commands

The figure below provides an additional example of the SITREX Automatic X.25 Simulator in action.

Further control over the automatic Simulator is affected through SPON, SPOF, SFON, and SFOF commands. These commands affect how SITREX handles incoming frames and packets. For example:

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>Chameleon</th>
<th>USER'S EQUIPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLON</td>
<td>SABM→</td>
<td>←UA</td>
</tr>
<tr>
<td></td>
<td><strong>Chameleon IN INFORMATION TRANSFER STATE</strong></td>
<td></td>
</tr>
<tr>
<td>PRST</td>
<td>RESTART→</td>
<td>←RESTART CONF.</td>
</tr>
<tr>
<td>RESTART</td>
<td></td>
<td>←RESTART</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESTART CONF.→</td>
</tr>
<tr>
<td>SPOF</td>
<td>←RESTART</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>←ETC...</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ETC...
8.3 SITREX MENUS

Introduction

This section describes the SITREX menus that are available for configuration. There are three SITREX menus:

- Physical Level Parameter Set-Up Menu
- Frame and Packet Level Set-Up Menu
- User Facilities Menu

The options in each of the SITREX menus are described in this section. General instructions for using set-up menus are in Section 2.2.

Each SITREX menu has an N (Next screen) or P (Previous screen) option so that you can switch between the three menus.

There is also a Save Parameters menu that you can access directly from any of the menus. The Save Parameters menu enables you to save and load parameter files and is described in Section 2.3.
Physical Level Menu

The SITREX Physical Level Set-Up menu enables you to change values for level one parameters. This menu is displayed below, with the valid values for each parameter.

![SITREX Physical Level Parameter Set-Up Menu](image)

** SITREX PHYSICAL LEVEL PARAMETER SET-UP **

A. Simulate . . . . . . . : DCE
B. Data Encoding . . . . : NRZ
C. Framing . . . . . . . : HDLC
D. Bit Rate . . . . . . . : 56000
E. Mode . . . . . . . . : Standby
F. User Facilities . . . : NO

SELECT: >

CHOICES:

INPUT: SELECT a letter
or type an option key shown below

Z=Run SITREX, ?=Help, S=Save parameters, ESC=Main Menu
N=Next Screen -> Set-up Frame and Packet level parameters

Figure 8.3-1: SITREX Physical Level Parameter Set-Up Menu

A. Simulate  
   DCE
   DTE

B. Data Encoding  
   NRZ
   NRZI

C. Framing  
   HDLC
   Xparent Bisync Displays additional menu option, G. Code, described below.

D. Bit Rate  
   50 to 64000
E. Mode

Standby  The Automatic Simulator waits on the line for incoming frames and packets, and responds accordingly. SITREX will not send outgoing frames or packets until it receives something.

Initiative  The Automatic Simulator tries to establish a link without any user input. In other words, it initiates the communication.

Manual  The Automatic Simulator will not respond to incoming frames or packets, nor will it send any outgoing frames or packets until you command it to do so. This is the same as using the SFOF (set frame level off) command.

F. User Facilities

NO  You cannot access the SITREX Facilities menu.

YES  You can access the SITREX Facilities menu to select the user facilities you want.

G. Code (Xparent
Bisync only)  ASCII/EVEN PARITY
              ASCII/ODD PARITY
              ASCII/NO PARITY
              EBCDIC
Frame & Packet Level Menu

Introduction

The SITREX Frame and Packet Level Set-up Menu enables you to change values for level two and three parameters. It is displayed when you press N for Next Screen from the Physical Level Parameter Set-up Menu.

** SITREX FRAME & PACKET LEVEL PARAMETER SET-UP **

A. Simulate ........... : NETWORK
B. Link Access ........... : LAPB
C. Frame Window ........... : 2
D. Frame Size N1 ........... : 1024
E. Frame Timer T1 ........... : 00100 m s
F. Re-transmissions N2 ........... : 10
G. Packet Window ........... : 2
H. Calling Address ........... : 12345
I. Called Address ........... : 54321
J. No. of SVCs ........... : 007
K. No. of PVCs ........... : 001
L. Sitrex Version ........... : CCITT

* * *

SELECT: >

CHOICES:

INPUT: SELECT a letter
or type an option key shown below

Z=Run SITREX, ?=Help, S=Save parameters, ESC=Main Menu
P=Previous Screen -> Set-up Physical Level Parameters, N = Next Screen -> Facilities

Figure 8.3-2: SITREX Frame & Packet Level Parameter Set-Up Menu

The choices for the SITREX Frame and Packet Level Parameter Set-up Menu are:

A. Simulate:

**Network** - If your system is configured as a Network, the automatic Simulator assigns the lowest available LCN (which you have specified as the number of PVCs), and increments from that point.

**Subscriber** - If your system is configured as a Subscriber, the automatic Simulator assigns the highest available LCN (which you have specified as the number of SVCs and PVCs), and decrements from that point.
B. Link Access: LAP
               LAPB

C. Frame Window: Range 1 - 3 frames

Note: Although the Chameleon screen prompt indicates that the acceptable range is 1 - 7 frames, SITREX will not function correctly if the frame window exceeds three frames.

D. Frame Length: Range 1 - 2040 bits

E. Frame Timer: Time to acknowledge frame (0 - 64,000 milliseconds)

F. Retransmissions: Range 1 - 99 Retransmissions

G. Packet Window: Range 1 - 7 packets

H. Calling Address: Address of Calling Station (Up to 15 digits)

I. Called Address: Address of Called Station (Up to 15 digits)

J. SVCs: Switched Virtual Circuits (SVCs) connect one location to many possible locations. (Range 0 - 255 circuits)

K. PVCs: Permanent Virtual Circuits (PVCs) are SVCs that have been assigned permanently between two points. (Range 0 - 255 circuits)

L. SITREX Version: CCITT
                   DATAPAC
                   NTT
SITREX Facilities Menu

Introduction

The SITREX Facilities menu (Figure 8.3-3) enables you to select the user facilities you need. The Facilities menu can be accessed only when the User Facilities parameter in the SITREX Physical Level menu is set to YES.

Each of the Facilities is described below. To select a facility, set the value to YES. When the fields are properly coded, the Automatic Simulator (at the packet level) accepts and responds to incoming calls with the appropriate facilities.

** SITREX FACILITIES **

A. Fast Select . . . . . . . . . : NO
B. Extended Packet Sequencing . . . : NO
C. Closed User Group Selection . . . : 
D. Packet Retransmission . . . . . . : NO
E. Flow Control Negotiation . . . . . : NO
F. Throughput Class Negotiation . . . : NO

SELECT> CHOICES:

INPUT: Select a letter
or type an option key shown below

Z = Run SITREX, ? = Help, S = Save parameters, ESC = Main Menu
P=Previous Screen --> Frame and Packet level parameters

---

Figure 8.3-3: SITREX Facilities Menu

A. Fast Select

This facility is a type of datagram service. Fast Select allows you to put data into a CALL packet and receive a CALL or CLEAR CONFIRMATION in return. SITREX supports incoming calls with data and turns the data field around on an outgoing call.

B. Extended Packet Sequencing

If you select this facility, calls received with extended sequencing will be accepted. All future calls accepted on that channel will have modulo 128 sequencing, as indicated by the GFI.
C. Closed User Group Selection

This field accepts a 2-digit or 4-digit decimal number or a blank space (none).

If you enter a value, only calls having this same CUG in the facility field as selected here are accepted. SITREX will accept only one CUG at a time.

If this field is blank, (no Closed User Groups - CUG’s), all CALL packets with the correct calling address are accepted.

D. Packet Retransmission

If YES, the subscriber can request the network to retransmit data packets. This is done by transmitting a packet REJ (reject) on the relevant Logical Channel Group Number and Logical Channel Number.

E. Flow Control Negotiation

If YES, this facility allows you to request packet length and window size on a call-by-call basis. This parameter is only relevant for incoming calls. For outgoing calls, use the SITREX Physical Parameter Set-up Menu.

F. Throughput Class Negotiation

If YES, SITREX accepts incoming calls with properly coded throughput facilities fields, but does not change the bit rate if acting as a network.

SITREX will verify that the incoming calls asking for Throughput Class Negotiation conform to the correct subscriber profile (refer to the CCITT specifications, Page 127, paragraph 7.4.2.6.2 - Table 17/X.25). This facility is not supported for packets transmitted by SITREX.
8.4 SITREX TRACE BUFFER

Introduction

The SITREX trace buffer stores and displays transmitted and received traffic from the line with control over frame and packet level interpretation. It captures all the traffic generated from the Automatic X.25 Simulator, a SITREX scenario, or traffic received from the line.

Section 8.2 describes how the trace buffer interacts with the Automatic X.25 Simulator and the command language for X.25 simulation. This section describes specific information about using the trace buffer, including:

- Activating and deactivating the trace function
- Displaying the trace contents
- Using trace commands to control what and how information is displayed
- Interpreting a trace display

Activating the Trace

The STON and STOF commands determine whether traffic is stored in the trace buffer. If the trace is on (STON), received and transmitted traffic is stored in the trace buffer. If the trace is off (STOF), traffic is not stored in the trace buffer. The default condition is for the trace buffer to be on. These commands can be used only at the ! command mode prompt.

If the trace is on (STON) you can control whether the contents of the buffer is displayed. To display the buffer you activate it, as follows:

1. Once you have selected SITREX simulation, the screen displays the message CHAMELEON SIMULATOR ACTIVE and the Trace banner is displayed (in pink) at the bottom of the screen.

2. You can only activate and deactivate the trace when you are in the Automatic X.25 Simulator mode at the blinking cursor. (You cannot activate the trace buffer from command mode at the ! prompt.)

3. To activate the trace function, type:

   PP <RETURN>

   and the message SITREX TRACE ACTIVE appears. Note that the first letter you type is shown as % on the SITREX screen.
4. To view the contents of the trace, you need to display the trace page, as follows:

a. Press SELECT until the Trace banner is highlighted.

b. Press the MOVE ↑ key several times until the page is displayed. (You can also press SHIFT + MOVE ↑ once to make the Trace page fill the entire screen.)

To hide the trace page, SELECT the Trace page and use the MOVE ↓ key several times until it is hidden. (If you used SHIFT + MOVE ↑ to display the page, you must use it again to restore the screen and hide the Trace page.)

5. While the SITREX trace is active, you can use special commands to control what is displayed on the Trace page. These commands are listed in the table below.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-9</td>
<td>Modifies the scrolling speed. Fastest = 1, Stop = 0</td>
</tr>
<tr>
<td>A</td>
<td>Toggles between ASCII and hex as format of displayed data.</td>
</tr>
<tr>
<td>F</td>
<td>Toggles display of frame level interpretation on/off.</td>
</tr>
<tr>
<td>P</td>
<td>Toggles display of packet level interpretation on/off.</td>
</tr>
<tr>
<td>R</td>
<td>Re-displays the contents of the trace.</td>
</tr>
<tr>
<td>CTRL C</td>
<td>Clears the trace memory.</td>
</tr>
<tr>
<td>CTRL P</td>
<td>Exits the trace mode and returns to the base Simulator level.</td>
</tr>
</tbody>
</table>

Table 8.4-1: Trace Commands

Commands F, A, and P are toggle commands. Press the key once to perform the action and again to discontinue the action.

For example, to display the Trace with packet level interpretation, but without frame level interpretation, you must type F, and then R.
6. To deactivate the trace so that data is no longer displayed, type:

   CTRL P

   The message SITREX TRACE IDLE appears on the Simulation page. You must be in the Trace Mode (and not at the command mode prompt !) to deactivate the trace function.

7. To turn the trace buffer off, so that data is not stored in the buffer, enter STOF at the command mode prompt (!).
This section illustrates a sample trace display and interprets five lines of the display.

The example, shown below, has the data displayed in hex, with both the frame and packet levels on. The lines that are interpreted on the following pages, are marked LINE A - E for easy reference.

**LINE A**

```
T * 002 01 P SABM
R * 002 01 P UA
```

**LINE B**

```
T * 006 01 I 0 0 /2000 RESTART 80
R * 005 03 I 0 1 /2000 RESTART C.
T * 002 03 RR 1
```

**LINE C**

```
T * 019 01 I 1 1 /2007 CALL 1234504 D 41 42 43 44 45 46
R * 013 03 I 1 2 /2007 CALL C. F 43 07 07 42 06 06
T * 002 03 RR 2
T * 007 01 I 2 2 /2007 CLEAR 84 00
R * 005 03 I 2 3 /2007 CLEAR C.
T * 002 03 RR 3
```

**LINE D**

```
T * 017 01 I 3 3 /2007 CALL 1234504 F 43 07 07 42 06 06
R * 007 03 I 3 4 /2007 CLEAR 03 42
T * 005 01 I 4 4 /2007 CLEAR C.
R * 002 01 RR 5
T * 010 01 I 5 4 /2007 CALL 12345
R * 013 03 I 4 6 /2007 CALL C. F 43 07 07 42 06 06
T * 002 03 RR 5
```

**LINE E**

```
T * 030 01 I 6 5 /E007 DA QD 000 W 000 48 45 4C 4F 20
        46 52 4F 40 20 50 53
R * 006 03 I 5 7 /6007 RR P. 001
T * 002 03 RR 6
T * 010 01 I 7 6 /E007 DA QD 001 000
R * 000 03 I 6 0 /6007 RR P. 002
T * 002 03 RR 7
T * 002 01 DISC / RD
R * 002 01 UA
```
**LINE A** indicates that a SABM has been transmitted.

```
T * 002 01 P SABM
```

- **T** = Transmit
- **R** = Receive
- **Good CRC = •**
- **Bad CRC = ?**
- Frame length in decimal.
- **Space =** correct length.
- \( > = \) Too long
- \( < = \) Too short

**LINE B** indicates that an I-Frame has been transmitted.

```
T * 006 01 I 0 0/2 0 00 RESTART 80
```

- Information frame.
- \( N(s) \) sequencing number in decimal.
- \( N(r) \) sequencing number in decimal.
- \( / \) indicates start of packet level.
- General Format Indicator (GFI).
- Indicates a cause code for the RESTART packet.
- Packet mnemonic.
- Logical Channel Number.
- Logical Channel Group Number (LCGN).
LINE C, contains user data in a Call packet, as shown below.

```
T* 019 01 1 1 /2007 CALL 1234504 D 41 42 43 44 45 46
```

- Transmitted
- Good CRC
- Frame Length
- Address
- I-Frame
- N(R)
- N(S)

Data in hex

D - Indicates data

Calling Address

Call Packet

GFI = Mod128
LGNC = 0
LCN = 07

/ = Packet Level

LINE D indicates a transmitted Call packet with facilities.

```
T* 017 01 1 3 3 /2007 CALL 1234504 F 43 07 07 42 06 06
```

Called and/or
Calling address,
Max. of 15 digits.

F indicates Facility.

Facility in HEX pairs.
LINE E indicates a transmitted data packet.

T * 030 01 16 5/E007 DA Q D 000 M 000 48 45 4C 4F 20 46 52

Indicates a data packet.

Q bit set.

D bit set.

P(r) Mod 128

Data in hex pairs or ASCII characters.

P(s) - Mod 128, (1 digit for Mod 8)

M bit set.

The example below shows what a Frame Reject would look like in the trace.

T * 005 03 CMDR/FRMR 3E 3 C 1 W X Y Z

Rejected control field in hex.

V(s)-rejected frame state variable.

Additional variables. (If the variable bit is set, the letter W - Z is displayed. If variable bit is not set, no letter is displayed.)

V(r)-rejected frame state variable.

C indicates a command; R indicates a response.
Inhibit Packet Level

The example below shows the trace display for the same events as shown on page 8.4-3. The difference is that the packet level interpretation has been inhibited. You use the trace commands listed on page 8.4-2 to control what is displayed.

To redisplay the trace in this format, you type P to toggle the packet level off, and then type R to redisplay.

```
T 002 01 SABM
R 002 01 UA
T 006 01 1 0 0 /20 00 FB 80
R 005 03 1 0 1 /20 00 FF
T 002 03 RR 1
T 019 01 1 1 1 /20 07 0B 07 12 34 50 40 41 42 43 4445
R 013 03 1 1 2 /20 07 0F 00 06 43 07 07 42 06 06
T 002 03 RR 2
T 007 01 1 2 2 /20 07 13 84 00
R 005 03 1 2 3 /20 07 07 17
T 002 03 RR 3
T 017 01 1 3 3 /20 07 0B 07 12 34 50 40 06 43 07 07 42 06 06
R 007 03 1 3 4 /20 07 13 03 42
T 005 01 1 4 4 /20 07 17
R 002 01 RR 5
T 010 01 1 5 4 /20 07 0B 05 12 34 50 00
R 013 03 1 4 6 /20 07 0F 00 06 43 07 07 42 06 06
T 002 03 RR 5
T 030 01 1 6 5 /E0 07 00 01 48 45 4C 4F 20 46 52 4F 20 46
52 4F 4D 20 50 53 45 55 44 4F 20 55
R 006 03 1 5 7 /E0 07 01 02
T 002 03 RR 6
etc.
```
8.5 SITREX COMMAND MODE

Introduction

SITREX command mode allows you to write and execute test scenarios that override the actions of the Automatic X.25 Simulator. The SITREX commands provide you with the control to:

- Send frames and packets
- Wait for specified events
- Set and change parameters
- Perform other protocol and error checking routines

Chapter 8.2 describes how the command language interacts with the Automatic X.25 Simulator and the Trace function. This section describes specific information about the command mode, including:

- Accessing command mode
- Command groups
- General syntax rules
- Controlling X.25 levels

SITREX is a highly flexible programming tool, designed specifically for X.25 Simulation. The large number of commands and the different ways of achieving the same result mean that care is needed when programming scenarios.

Line protocols, in general, are time critical, and SITREX commands take time to execute. Therefore, long print statements and excessive jumping should be avoided.

SELESE and WAIT commands should be separated by at least one command line. Also, remarks (REM statements) take up considerable space in the 4K bytes allocated to SITREX scenario memory.

Command Mode Access

To access the SITREX command mode, you must first select SITREX simulation. If you do not know how to do this, refer to Section 2.1 for instructions.

Once you have selected SITREX simulation, the screen displays the message CHAMELEON SIMULATOR ACTIVE and the Trace banner is displayed at the bottom of the screen.
Note that the first letter you type is shown as % on the SITREX screen. To exit the base Simulator mode and enter command mode, type:

```
PS  <RETURN>
```

The following message and the ! prompt appear:

```
Chameleon X.25 SIMULATION
SITREX VERSION X.XX
!
```

**Command Groups**

SITREX commands are divided into three major groups:

- Low-Level Commands
- Mid-Level Commands
- High-Level Commands

These three groups are described below, with an example of each.

**Low-Level Commands**

Low-level commands transmit individual frames or packets. These commands are independent of the Automatic X.25 Simulator.

For example, the command `FH012F`, sends a user-defined frame in hex, where `012F` is the frame address. This command does not expect a response.

**Mid-Level Commands**

Mid-level commands allow you to use mnemonics to transmit frames or packets. Additionally, SITREX provides missing information, such as the frame address.

For example, the command `FSABM` sends a `SABM` at the frame level. SITREX supplies the frame address, depending upon your position as a Subscriber or a Network. This command does not expect a response.
High-Level Commands

High-level commands are instructions to the Automatic X.25 Simulator to perform certain actions and wait for proper responses.

For example, the SLON command Set Link ON. SLON tells the Automatic X.25 Simulator to go to an Information transfer state. At this point, the automatic X.25 Simulator transmits a SABM with the primary address. It then decrements N(2) and starts the T1 timer.

The Automatic X.25 Simulator expects a UA. If there is no response within the T1 timeout, the Automatic X.25 Simulator looks at N(2). If it is not 0, it sends another SABM, otherwise SITREX automatically sets the link off.

General Syntax Rules

The following are rules to follow when using SITREX commands. The specific syntax of the commands is described in Chapter 8.7.

1. The maximum size of a SITREX program is 4K bytes.

2. A command line is a chain of one or more commands terminated by a carriage return. The most common structure of a command is:

   (LINE NUMBER) COMMAND (VARIABLES): < RETURN >

   Additional spaces shown here in the syntax of a command are for clarity only, and should not be entered.

3. Commands can be followed by variables and terminated by a colon (:) or carriage return. Do not use a colon in a PRINT command, because the colon will be taken literally and printed.

4. If more than one command is used in a line, delimit the commands with a colon ( : ).

5. Commands can be entered in upper or lower case letters.

6. Variables are separated from commands by one space.

7. A command can include optional parameters or fields. In the syntax descriptions, optional parameters are shown in parentheses ( ), but the parentheses should not be entered.
8. The maximum size of a command line is 255 characters, including the line number, spaces, and carriage returns.

9. The following rules apply to line numbers:
   - Command lines used in programs (statement commands) must have line numbers.
   - Commands executed immediately at the ! prompt (direct commands) do not have line numbers.
   - A line number is a decimal number between 1 and 9999 that is the first entry on a command line.
   - A single blank space follows a line number.

10. Commands of the same group generally have one letter in common. For example, FDISC and FSABM are in the Frame Level group.

11. You should carefully determine that the commands in your program are coherent and consistent. For example, the PRST (Packet Restart) command has no effect unless a link has been established.

12. N(s), N(r), P(s), P(r), V(s), V(r), F, M, P, Q, W, X, Y, and Z represent the conventional parameters of the X.25 protocol. When these parameters appear in parentheses ( ) in the command syntax, they are optional. If they are omitted, their values are automatically determined by the Automatic X.25 Simulator, depending on the status of the link.

13. In the syntax description, the following conventions apply:
   - b = Enter binary digits. Binary strings are split into 8 bit groups to be converted into bytes.
   - h = Enter hexadecimal characters. Hex strings are split into groups of 2 characters.
   - a = Enter printable ASCII characters
     - Lower order digits are on the right
     - A space, if present, indicates the end of a byte
     - The preceding binary digit (or hex character) is the lower order digit of the preceding byte
     - Missing elements are assumed to be all zeros
Substitution

Some higher-level commands have optional fields that are lower-level commands. In other words, you can have a command within a command.

For example, the WAIT command synchronizes a test scenario to wait for the arrival of a specific frame. One form of syntax for the WAIT command is listed as:

\[ W(\text{FRAME}) \]

The Wait command is the letter W, optionally followed by a Frame Level command. (Frame Level commands are one of the functional groups of commands.) If you wanted to wait for an I-frame, you could use FI (a frame level command). The FI command, takes a packet level command, as indicated by the syntax description of the FI command:

\[ \text{FI(PACKET)} \]

The optional field PACKET can be replaced by any of the Packet Level commands. For example, you could indicate a Restart Packet with the PRST command, which uses the following syntax:

\[ \text{PRST(h1h2)(h3h4)} \]

In the PRST syntax, h1h2 is an optional 2-digit hex cause code and h3h4 is an optional 2-digit hex diagnostic code.

Using this process of substitution, if you wanted to enter a command that waits until it receives an I-frame containing a Restart packet with a cause code of 34, you would enter:

\[ W\text{FI}PRST34 \]

\[ W = \text{Wait for the following item.} \]
\[ \text{FI = I-Frame} \]
\[ \text{PRST34 = Restart Packet with cause code 34} \]
Syntax Errors

When SITREX detects a violation of the syntax rules, the line is re-displayed. The cursor stops before the first illegal character, and SITREX emits a beep.

You must type the correct characters following the error detected. The word error is not displayed. If the first character is incorrect, or not recognized, only the cursor is re-displayed.

For example, if you entered a line number and a command, but omitted the space between them, such as:

100SLON <RETURN>

This command is displayed as

100

The line number must be be followed by a blank space.
8.6 SITREX COMMAND INDEX

Introduction
The SITREX programming commands are listed in Section 8.7, grouped by function. To help you locate a specific command in Section 8.7, this section contains two indexes:

- Alphabetical Command Index
- Functional Command Index

If you know the name of a command, you can use the Alphabetical Command Index to determine which page in Section 8.7 contains the complete command description.

If you need a command that performs a particular function, but you do not know which command to use, refer to the Functional Command Index to locate the command.

Alphabetical Index
The Alphabetical Command Index begins on the following page. It provides the command name, a brief description of its function, and a page reference.

The complete syntax is not listed in this index. The only part of the command which is shown in this table, is that part which uniquely identifies it. Where additional information is required by the syntax, an ellipsis is inserted. For example, the entry:

FA...

indicates that additional letters or fields (optional or required) will follow the letters FA. Ellipses may be shown in one or more places within a single command. For a complete description of the syntax, you must refer to the indicated page in Section 8.7.

Many frame level commands can include an optional Poll or Final bit, which is shown in the index as (F) or (P). For example:

F(P)DISC

indicates that this command (which sends a Disconnect frame), may be entered as:

FDISC or FPDISC

depending on whether the poll bit is to be set.
**TABLE 8.6-1: SITREX ALPHABETICAL COMMAND INDEX**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>SEE PAGE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;...</td>
<td>8.7-49</td>
<td>Chains programs.</td>
</tr>
<tr>
<td>*...</td>
<td>8.7-44</td>
<td>* begins a loop.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...Command lines executed within loop.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; ends the loop.</td>
</tr>
<tr>
<td>DISP(X)</td>
<td>8.7-75</td>
<td>Displays values of timers, counters, variables or contents and length of message buffer.</td>
</tr>
<tr>
<td>EXIT</td>
<td>8.7-2</td>
<td>Exits command mode and return to the Automatic X.25 Simulator.</td>
</tr>
<tr>
<td>FA...</td>
<td>8.7-2</td>
<td>Sends user-defined frame in ASCII.</td>
</tr>
<tr>
<td>FB...</td>
<td>8.7-2</td>
<td>Sends user-defined frame in binary.</td>
</tr>
<tr>
<td>F(F)CMDR...</td>
<td>8.7-4</td>
<td>Sends a CMDR frame.</td>
</tr>
<tr>
<td>F(P)DISC</td>
<td>8.7-3</td>
<td>Sends DISC frame.</td>
</tr>
<tr>
<td>F(F)DM</td>
<td>8.7-4</td>
<td>Sends a DM frame.</td>
</tr>
<tr>
<td>FH...</td>
<td>8.7-2</td>
<td>Sends user-defined frame in hex.</td>
</tr>
<tr>
<td>F(P)I...</td>
<td>8.7-7</td>
<td>Sends an I-frame.</td>
</tr>
<tr>
<td>FM</td>
<td>8.7-65</td>
<td>Transmits the message buffer.</td>
</tr>
<tr>
<td>FPREJ...</td>
<td>8.7-5</td>
<td>Sends supervisory polled command REJ frame.</td>
</tr>
<tr>
<td>FPRNR...</td>
<td>8.7-5</td>
<td>Sends supervisory polled command RNR frame.</td>
</tr>
<tr>
<td>FPRR...</td>
<td>8.7-5</td>
<td>Sends supervisory polled command RR frame.</td>
</tr>
<tr>
<td>F(F)REJ...</td>
<td>8.7-6</td>
<td>Sends a REJ frame.</td>
</tr>
<tr>
<td>F(F)RNR...</td>
<td>8.7-6</td>
<td>Sends a RNR response frame.</td>
</tr>
<tr>
<td>F(F)RR...</td>
<td>8.7-6</td>
<td>Sends a RR response frame.</td>
</tr>
<tr>
<td>F(P)SABM</td>
<td>8.7-3</td>
<td>Sends SABM on primary address</td>
</tr>
<tr>
<td>F(F)UA</td>
<td>8.7-4</td>
<td>Sends an UA frame.</td>
</tr>
<tr>
<td>GOSUB</td>
<td>8.7-46</td>
<td>Unconditionally jumps to specified subroutine.</td>
</tr>
<tr>
<td>GOTO...</td>
<td>8.7-46</td>
<td>Unconditionally jumps to specified line.</td>
</tr>
<tr>
<td>HALT</td>
<td>8.7-47</td>
<td>Returns to Chameleon main menu.</td>
</tr>
<tr>
<td>IF1... ±</td>
<td>8.7-29</td>
<td>Tests specified lead for mark or space and jumps to specified line when true.</td>
</tr>
<tr>
<td>IFC...</td>
<td>8.7-45</td>
<td>Tests counter and jumps to specified line when true.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>SEE PAGE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IFT...</td>
<td>8.7-45</td>
<td>Tests timer and jumps to specified line when true.</td>
</tr>
<tr>
<td>INPUT X...</td>
<td>8.7-58</td>
<td>Waits for user input and stores it to a variable.</td>
</tr>
<tr>
<td>IS...</td>
<td>8.7-64</td>
<td>Compares message buffer to specified byte and mask configuration. If true, jumps to specified line.</td>
</tr>
<tr>
<td>IX...</td>
<td>8.7-59</td>
<td>Tests a variable for a value. If true, jumps to specified line number.</td>
</tr>
<tr>
<td>LDISP(X)</td>
<td>8.7-75</td>
<td>Outputs timers, counters, variables or contents of message buffer to a remote terminal or printer.</td>
</tr>
<tr>
<td>LIST</td>
<td>8.7-78</td>
<td>Displays the program lines in memory.</td>
</tr>
<tr>
<td>LLIST</td>
<td>8.7-79</td>
<td>Outputs the program lines in memory to a remote terminal or printer.</td>
</tr>
<tr>
<td>LNKUP</td>
<td>8.7-21</td>
<td>Forces Simulator to assume that the link is up.</td>
</tr>
<tr>
<td>LOAD</td>
<td>8.7-50</td>
<td>Loads a program file into memory.</td>
</tr>
<tr>
<td>LPRINT</td>
<td>8.7-80</td>
<td>Outputs a user-specified message to a printer or remote terminal.</td>
</tr>
<tr>
<td>LTPRINT</td>
<td>8.7-69</td>
<td>Outputs the contents of the trace buffer to a printer or remote terminal.</td>
</tr>
<tr>
<td>NEW</td>
<td>8.7-52</td>
<td>Erases current program in memory.</td>
</tr>
<tr>
<td>PA...</td>
<td>8.7-9</td>
<td>Sends a user-defined packet in ASCII.</td>
</tr>
<tr>
<td>PCRST</td>
<td>8.7-12</td>
<td>Sends a Restart Confirmation packet.</td>
</tr>
<tr>
<td>PDIAG...</td>
<td>8.7-15</td>
<td>Sends a Diagnostic packet.</td>
</tr>
<tr>
<td>PH...</td>
<td>8.7-9</td>
<td>Sends a user-defined packet in hex.</td>
</tr>
<tr>
<td>PM</td>
<td>8.7-65</td>
<td>Transmits the frame, assigning the contents of the message buffer to the l-field.</td>
</tr>
<tr>
<td>PRINT</td>
<td>8.7-77</td>
<td>Displays a user-specified message to the screen.</td>
</tr>
<tr>
<td>PRST...</td>
<td>8.7-12</td>
<td>Sends a Restart packet.</td>
</tr>
<tr>
<td>PU...CALL</td>
<td>8.7-10</td>
<td>Sends a Call packet.</td>
</tr>
<tr>
<td>PU...CCALL</td>
<td>8.7-10</td>
<td>Sends a Call Confirmation packet.</td>
</tr>
<tr>
<td>PU...CCLEAR</td>
<td>8.7-13</td>
<td>Sends a Clear Confirmation packet.</td>
</tr>
<tr>
<td>PU...CINT</td>
<td>8.7-13</td>
<td>Sends an Interrupt Confirmation packet.</td>
</tr>
</tbody>
</table>
### TABLE 8.6-1: SITREX ALPHABETICAL COMMAND INDEX (CONT.)

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>SEE PAGE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU...CLEAR...</td>
<td>8.7-13</td>
<td>Sends a Clear packet.</td>
</tr>
<tr>
<td>PU...CRSET...</td>
<td>8.7-13</td>
<td>Sends a Reset Confirmation packet.</td>
</tr>
<tr>
<td>PU...D...</td>
<td>8.7-14</td>
<td>Sends a data packet.</td>
</tr>
<tr>
<td>PU...DS</td>
<td>8.7-65</td>
<td>Transmits a frame, assigning the contents of the message buffer to the data portion of the I-field, from a specified pseudo-user.</td>
</tr>
<tr>
<td>PU...INT...</td>
<td>8.7-13</td>
<td>Sends an interrupt packet.</td>
</tr>
<tr>
<td>PU...REJ...</td>
<td>8.7-11</td>
<td>Sends a REJ packet.</td>
</tr>
<tr>
<td>PU...RNR...</td>
<td>8.7-11</td>
<td>Sends a RNR packet.</td>
</tr>
<tr>
<td>PU...RR...</td>
<td>8.7-11</td>
<td>Sends a RR packet.</td>
</tr>
<tr>
<td>PU...RSET</td>
<td>8.7-13</td>
<td>Sends a Reset packet.</td>
</tr>
<tr>
<td>REM</td>
<td>8.7-51</td>
<td>Programmer’s remark.</td>
</tr>
<tr>
<td>RETURN</td>
<td>8.7-46</td>
<td>Returns control from a subroutine called by a GOSUB, to the command following the GOSUB command.</td>
</tr>
<tr>
<td>RUN</td>
<td>8.7-53</td>
<td>Executes the program in memory.</td>
</tr>
<tr>
<td>S1...±</td>
<td>8.7-28</td>
<td>Sets specified interface lead to space (+) or mark (-).</td>
</tr>
<tr>
<td>SADRWT...</td>
<td>8.7-41</td>
<td>Sets jump address for Watch-dog timer.</td>
</tr>
<tr>
<td>SAVE</td>
<td>8.7-54</td>
<td>Saves the program in memory to disk.</td>
</tr>
<tr>
<td>SC...</td>
<td>8.7-31</td>
<td>Increments, decrements or sets user-defined counter.</td>
</tr>
<tr>
<td>SCRC±</td>
<td>8.7-22</td>
<td>Specifies that frames include good (+) or no (-) CRC.</td>
</tr>
<tr>
<td>SELSE...</td>
<td>8.7-41</td>
<td>Sets jump address for WAIT command.</td>
</tr>
<tr>
<td>SFOF</td>
<td>8.7-17</td>
<td>Sets frame level off.</td>
</tr>
<tr>
<td>SFON</td>
<td>8.7-17</td>
<td>Sets frame level on.</td>
</tr>
<tr>
<td>SGT...</td>
<td>8.7-25</td>
<td>Sets length of data packet sent by a traffic generator.</td>
</tr>
<tr>
<td>SK...</td>
<td>8.7-24</td>
<td>Sets frame window size.</td>
</tr>
<tr>
<td>SLG...</td>
<td>8.7-27</td>
<td>Sets the Logical Channel Group Number for next call.</td>
</tr>
<tr>
<td>SLOF</td>
<td>8.7-20</td>
<td>Sets the link off.</td>
</tr>
<tr>
<td>SLOM</td>
<td>8.7-20</td>
<td>Sets the link on.</td>
</tr>
<tr>
<td>SM...</td>
<td>8.7-60</td>
<td>Writes the specified values into the message buffer.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>SEE PAGE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SNR(±)</td>
<td>8.7-26</td>
<td>Increments (+) or decrements (-) N(r).</td>
</tr>
<tr>
<td>SNS(±)</td>
<td>8.7-26</td>
<td>Increments (+), decrements, or sets value of N(s).</td>
</tr>
<tr>
<td>SPA...</td>
<td>8.7-23</td>
<td>Sets Primary address in hex.</td>
</tr>
<tr>
<td>SPOF</td>
<td>8.7-19</td>
<td>Sets packet level off.</td>
</tr>
<tr>
<td>SPON</td>
<td>8.7-19</td>
<td>Sets packet level on.</td>
</tr>
<tr>
<td>SPU...</td>
<td>8.7-32</td>
<td>Sets or alters attributes of specified pseudo-user.</td>
</tr>
<tr>
<td>SSA...</td>
<td>8.7-23</td>
<td>Sets Secondary address in hex.</td>
</tr>
<tr>
<td>ST...</td>
<td>8.7-30</td>
<td>Sets timer T', T&quot; or user-defined timer.</td>
</tr>
<tr>
<td>STOF</td>
<td>8.7-71</td>
<td>Sets the trace off (idle).</td>
</tr>
<tr>
<td>STON</td>
<td>8.7-71</td>
<td>Sets the trace on (active).</td>
</tr>
<tr>
<td>STR...</td>
<td>8.7-73</td>
<td>Sets number of data bytes displayed by trace.</td>
</tr>
<tr>
<td>STU...</td>
<td>8.7-30</td>
<td>Sets timer TU.</td>
</tr>
<tr>
<td>SU...LR</td>
<td>8.7-25</td>
<td>Sets maximum length of received data packet.</td>
</tr>
<tr>
<td>SU...LT</td>
<td>8.7-25</td>
<td>Sets maximum length of transmitted data packet.</td>
</tr>
<tr>
<td>SU...PR(±)</td>
<td>8.7-26</td>
<td>Increments (+), decrements, (-) or sets value of P(r).</td>
</tr>
<tr>
<td>SU...PS(±)</td>
<td>8.7-26</td>
<td>Increments (+), decrements (-), or sets value of P(s).</td>
</tr>
<tr>
<td>SU...VC...</td>
<td>8.7-33</td>
<td>Set up a Switched Virtual Circuit (SVC).</td>
</tr>
<tr>
<td>SU...VP...</td>
<td>8.7-33</td>
<td>Set up a Permanent Virtual Circuit (PVC).</td>
</tr>
<tr>
<td>SU...WR...</td>
<td>8.7-24</td>
<td>Sets receive packet window size.</td>
</tr>
<tr>
<td>SU...WT...</td>
<td>8.7-24</td>
<td>Sets transmit packet window size.</td>
</tr>
<tr>
<td>SWT...</td>
<td>8.7-42</td>
<td>Sets the Watch-Dog timer.</td>
</tr>
<tr>
<td>SX...</td>
<td>8.7-56</td>
<td>Assigns a value to a numeric variable using arithmetic or logical operations.</td>
</tr>
<tr>
<td>SX...D...</td>
<td>8.7-57</td>
<td>Shifts the contents of the variable to the right (D).</td>
</tr>
<tr>
<td>SX...G...</td>
<td>8.7-57</td>
<td>Shifts the contents of the variable to the left (G).</td>
</tr>
<tr>
<td>SX...I...</td>
<td>8.7-60</td>
<td>Inserts a value in the message buffer at a specified location.</td>
</tr>
<tr>
<td>SX...I00</td>
<td>8.7-63</td>
<td>Extracts message buffer length.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>SEE PAGE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>SX...L...</td>
<td>8.7-57</td>
<td>Rotates the contents of the variable to the left.</td>
</tr>
<tr>
<td>SX...O...</td>
<td>8.7-62</td>
<td>Stores a specified byte from the message buffer in a variable.</td>
</tr>
<tr>
<td>SX...000</td>
<td>8.7-63</td>
<td>Sets message buffer length.</td>
</tr>
<tr>
<td>SX...R...</td>
<td>8.7-57</td>
<td>Rotates the contents of the variable to the right.</td>
</tr>
<tr>
<td>TLOAD</td>
<td>8.7-68</td>
<td>Loads a trace file from disk to memory.</td>
</tr>
<tr>
<td>TSAVE</td>
<td>8.7-70</td>
<td>Save the trace file in memory to disk.</td>
</tr>
<tr>
<td>TPRINT</td>
<td>8.7-67</td>
<td>Displays the contents of the trace buffer.</td>
</tr>
<tr>
<td>TRACE</td>
<td>8.7-72</td>
<td>Clears the trace buffer.</td>
</tr>
<tr>
<td>WF...</td>
<td>8.7-36</td>
<td>Waits for a specific frame before continuing scenario.</td>
</tr>
<tr>
<td>WP...</td>
<td>8.7-36</td>
<td>Waits for a specific packet before continuing scenario.</td>
</tr>
<tr>
<td>WSF...</td>
<td>8.7-36</td>
<td>Waits for a specific frame stores it in the message buffer.</td>
</tr>
<tr>
<td>WSP...</td>
<td>8.7-36</td>
<td>Waits for a specific packet and stores it in the message buffer.</td>
</tr>
<tr>
<td>WST</td>
<td>8.7-38</td>
<td>Waits for a specific byte mask configuration and stores it in the message buffer.</td>
</tr>
<tr>
<td>WT...</td>
<td>8.7-38</td>
<td>Waits for a specific byte mask and jumps to specified address if not received.</td>
</tr>
</tbody>
</table>
The Functional Command index lists the SITREX command in these functional groups:

- Frame Level
- Packet Level
- Parameter
- Wait
- Loop, Jump, and Reinitialization
- Program Management
- Numeric Variable
- Message Buffer
- Trace Buffer
- Display and Print

Frame Level
This group of command transmits any type of HDLC frame.

Packet Level
This group of commands transmits any type of X.25 packet.

Parameter
Set commands allow you to define the value of X.25 global parameters, such as frame window size.

Wait
Wait Command enable you to wait for a specific type of frame, packet, or byte mask. The timers associated with the WAIT commands are also included in this group.

Loop, Jump and Reinitialization
These commands enable you to jump to a specified line in your scenario, or set up a loop. The reinitialization commands enable you to exit from SITREX, access parameter set-up menus, or stop program execution.

Program Management
This group includes loading, running, saving, chaining, listing and printing scenario files.

Numeric Variable
This group enables you to initialize and manipulate numeric variables.

Message Buffer
This group includes commands that define messages for the buffer, transmit messages, and extract information about the data in the message buffer.

Trace Buffer
This group allows you to manipulate the trace buffer.

Display and Print Commands
This group includes commands that display information to the screen or output the information to a printer or remote device.
### TABLE 8.6-2: SITREX FRAME LEVEL COMMANDS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Defined Frames</td>
<td>8.7-2</td>
</tr>
<tr>
<td>Unnumbered Command Frames</td>
<td>8.7-3</td>
</tr>
<tr>
<td>Unnumbered Response Frames</td>
<td>8.7-4</td>
</tr>
<tr>
<td>Numbered Command Frames</td>
<td>8.7-5</td>
</tr>
<tr>
<td>Numbered Response Frames</td>
<td>8.7-6</td>
</tr>
<tr>
<td>Information Frames</td>
<td>8.7-7</td>
</tr>
</tbody>
</table>

### TABLE 8.6-3: SITREX PACKET LEVEL COMMANDS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Defined Packets</td>
<td>8.7-9</td>
</tr>
<tr>
<td>Call Packets</td>
<td>8.7-10</td>
</tr>
<tr>
<td>Supervisory Packets</td>
<td>8.7-11</td>
</tr>
<tr>
<td>Restart/Restart Confirmation Packets</td>
<td>8.7-12</td>
</tr>
<tr>
<td>Clear/Reset/Interrupt &amp; Confirmation</td>
<td>8.7-13</td>
</tr>
<tr>
<td>Data Packets</td>
<td>8.7-14</td>
</tr>
<tr>
<td>Diagnostic Packets</td>
<td>8.7-15</td>
</tr>
</tbody>
</table>
### TABLE 8.6-4: SITREX PARAMETER COMMANDS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Frame Level On/Off</td>
<td>8.7-17</td>
</tr>
<tr>
<td>Set Packet Level On/Off</td>
<td>8.7-19</td>
</tr>
<tr>
<td>Set Link On/Off</td>
<td>8.7-20</td>
</tr>
<tr>
<td>Force Link On</td>
<td>8.7-21</td>
</tr>
<tr>
<td>Transmit CRC On/Off</td>
<td>8.7-22</td>
</tr>
<tr>
<td>Set Primary/Secondary Address</td>
<td>8.7-23</td>
</tr>
<tr>
<td>Set Frame/Packet Windows</td>
<td>8.7-24</td>
</tr>
<tr>
<td>Set Data Packet Length</td>
<td>8.7-25</td>
</tr>
<tr>
<td>Set Frame/Packet State Variables</td>
<td>8.7-26</td>
</tr>
<tr>
<td>Set Logical Channel Group Number</td>
<td>8.7-27</td>
</tr>
<tr>
<td>Set Interface Leads</td>
<td>8.7-28</td>
</tr>
<tr>
<td>Test Interface Leads</td>
<td>8.7-29</td>
</tr>
<tr>
<td>Set Timers</td>
<td>8.7-30</td>
</tr>
<tr>
<td>Set Counters</td>
<td>8.7-31</td>
</tr>
<tr>
<td>Set Pseudo-User Type</td>
<td>8.7-32</td>
</tr>
<tr>
<td>Set PVCs and SVCs</td>
<td>8.7-33</td>
</tr>
</tbody>
</table>

### TABLE 8.6-5: SITREX WAIT COMMANDS

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for Frame or Packet</td>
<td>8.7-36</td>
</tr>
<tr>
<td>Wait for Byte Mask</td>
<td>8.7-38</td>
</tr>
<tr>
<td>Set Jump Addresses for Wait Commands</td>
<td>8.7-41</td>
</tr>
</tbody>
</table>
**TABLE 8.6-6: SITREX LOOP, JUMP AND REINITIALIZATION COMMANDS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up Program Loops</td>
<td>8.7-44</td>
</tr>
<tr>
<td>Conditional Jump (IF)</td>
<td>8.7-45</td>
</tr>
<tr>
<td>Unconditional Jump</td>
<td>8.7-46</td>
</tr>
<tr>
<td>Reinitialization</td>
<td>8.7-47</td>
</tr>
</tbody>
</table>

**TABLE 8.6-7: SITREX PROGRAM MANAGEMENT COMMANDS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain Programs</td>
<td>8.7-49</td>
</tr>
<tr>
<td>Save Program</td>
<td>8.7-54</td>
</tr>
<tr>
<td>Load Program</td>
<td>8.7-50</td>
</tr>
<tr>
<td>Remarks</td>
<td>8.7-51</td>
</tr>
<tr>
<td>New Program</td>
<td>8.7-52</td>
</tr>
<tr>
<td>Run Program</td>
<td>8.7-53</td>
</tr>
</tbody>
</table>

**TABLE 8.6-8: SITREX NUMERIC VARIABLE COMMANDS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Operations</td>
<td>8.7-56</td>
</tr>
<tr>
<td>Shift and Rotate</td>
<td>8.7-57</td>
</tr>
<tr>
<td>Keyboard Input</td>
<td>8.7-58</td>
</tr>
<tr>
<td>Test Variables</td>
<td>8.7-59</td>
</tr>
</tbody>
</table>
**TABLE 8.6-9: SITREX MESSAGE BUFFER COMMANDS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining Messages</td>
<td>8.7-60</td>
</tr>
<tr>
<td>Byte Extraction</td>
<td>8.7-62</td>
</tr>
<tr>
<td>Assign Buffer Length</td>
<td>8.7-63</td>
</tr>
<tr>
<td>Test Message Buffer Contents</td>
<td>8.7-64</td>
</tr>
<tr>
<td>Wait and Store in Buffer</td>
<td>8.7-36</td>
</tr>
<tr>
<td>Transmit Message</td>
<td>8.7-65</td>
</tr>
</tbody>
</table>

**TABLE 8.6-10: SITREX TRACE BUFFER COMMANDS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display contents</td>
<td>8.7-67</td>
</tr>
<tr>
<td>Load Trace file</td>
<td>8.7-68</td>
</tr>
<tr>
<td>Print contents of trace buffer</td>
<td>8.7-69</td>
</tr>
<tr>
<td>Save Trace file</td>
<td>8.7-70</td>
</tr>
<tr>
<td>Set Trace On/Off</td>
<td>8.7-71</td>
</tr>
<tr>
<td>Clear Trace Buffer</td>
<td>8.7-72</td>
</tr>
<tr>
<td>Set Trace Length</td>
<td>8.7-73</td>
</tr>
</tbody>
</table>

**TABLE 8.6-11: SITREX DISPLAY AND PRINT COMMANDS**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display contents of trace buffer</td>
<td>8.7-67</td>
</tr>
<tr>
<td>Print contents of trace buffer</td>
<td>8.7-69</td>
</tr>
<tr>
<td>Display User-parameters</td>
<td>8.7-75</td>
</tr>
<tr>
<td>Display Screen Messages &amp; Prompts</td>
<td>8.7-77</td>
</tr>
<tr>
<td>List Program File</td>
<td>8.7-78</td>
</tr>
<tr>
<td>Print Program File</td>
<td>8.7-79</td>
</tr>
<tr>
<td>Print Screen Messages &amp; Prompts</td>
<td>8.7-80</td>
</tr>
</tbody>
</table>
FRAME LEVEL COMMANDS

Introduction

Frame level commands allow you to transmit any type of HDLC frame. This enables you to bypass the Automatic X.25 Simulator with specific frame commands. Refer to Section 8.2 for a description of how the frame level commands interact with the Automatic X.25 Simulator.

Examples:

The command FUA tells SITREX to send an unnumbered acknowledgment.

The command FPRR3 sends a polled RR with an N(r) equal to 3 on a transmitted frame.

Poll/Final Bit

The following conventions apply to the frame Level commands:

- (P) indicates that an optional poll bit can be set and the command sent on a primary address. The primary address is the address the particular unit, whether Network or Subscriber, uses to send a command.

- (F) indicates that an optional final bit can be set and the response sent on a secondary address. The secondary address refers to the address the unit uses to send a response.
Send User-Defined Frame

Description  These commands send a user-defined frame in binary, ASCII, or hex.

Type  Statement or Direct

Syntax  
- **FBb**........**b**  Sends frame defined in binary. Multiple spaces are interpreted as one space. Enter data in 8-bit bytes, separated by spaces.
- **FAa**........**a**  Sends frame defined in ASCII.
- **FHh**........**h**  Sends frame defined in hexadecimal. Multiple spaces are interpreted as one space.

You can combine ASCII and hex in the same line. To switch between ASCII and hex, press the Escape key. The Chameleon display a tilde (\~) each time you press the Escape key.

Example 1  In this example, ASCII data is placed in the data field of a data packet. The corresponding frame is transmitted whether it is valid or invalid.

```
FH0100100000~THIS PART IS IN ASCII
```

Example 2  This example sends a frame with data in binary. When using binary, enter data in 8-bit bytes, separated by spaces.

```
FB00000001 00011111
```
### Send Unnumbered Commands

**Description**
These commands send either a DISC or SABM.

**Type**
Statement or Direct

**Syntax**
- F(P)DISC: Sends a polled or unpolled DISC on the primary address.
- F(P)SABM: Sends a polled or unpolled SABM on the primary address.

**Example 1**
This example sends an unpolled disconnect command frame.

```
FDISC
```

**Example 2**
This example sends a polled SABM command frame.

```
FPSABM
```
### Send Unnumbered Responses

**Description**
These commands send either a UA, DM or CMDR frame.

**Type**
Statement or Direct

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(F)UA</td>
<td>Sends a UA frame.</td>
</tr>
<tr>
<td>F(F)DM</td>
<td>Sends a DM frame.</td>
</tr>
<tr>
<td>F(F)CMDRh1h2(Vs)(Vs)(Vr)(B)(W)(X)(Y)(Z)</td>
<td>Sends a CMDR frame.</td>
</tr>
</tbody>
</table>

**CMDR** is followed by two hex characters h1 h2, which are the control fields of the rejected frame.

Vs is the Ns of the last valid frame received. Vr is the Nr of the last valid frame received. Vs and Vr vary from 0 to 7.

B is the Command/Response bit (C/R) of the frame. It is set to 1 if rejecting a response, and set to 0 if rejecting a command.

W, X, Y, Z represent the 17th to the 20th bits, respectively, of the CMDR frame according to the X.25 protocol.

B, W, X, Y, Z are literals. If these letters are entered in a command, the corresponding bit is set to 1. If they are not entered, the bit is set to 0.

**Example 1**
This example sends a CMDR frame with a rejected control field equal to 34, Vs unspecified, Vr = 3, W and Z equal to 1, and X and Y equal to 0.

`FCMDR 34,3WZ`

**Example 2**
This example sends a Command Reject with the final bit set (the second F in the command specifies the final bit). The control field of the rejected frame = 3E, Vs = 1, Vr = 2, and the literals BWXY and Z are all set to 1.

`FFCMDR 3E1,2BWXYZ`
## Send Numbered Commands

<table>
<thead>
<tr>
<th>Description</th>
<th>These commands send the supervisory polled command frames: Receiver Ready (RR), Receiver Not Ready (RNR), or Frame Reject (REJ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note:</td>
<td>To send unpolled supervisory command frames, refer to the commands FH, FB, SPA and SSA.</td>
</tr>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>FPRR(Nr) Sends the supervisory polled command frame RR.</td>
</tr>
<tr>
<td></td>
<td>FPRNR(Nr) Sends the supervisory polled command frame RNR.</td>
</tr>
<tr>
<td></td>
<td>FPREJ(Nr) Sends the supervisory polled command frame REJ.</td>
</tr>
<tr>
<td></td>
<td>where: N is in the range 0 to 7.</td>
</tr>
<tr>
<td></td>
<td>Since this is a command, and not a response, the poll bit is required. If P is omitted, SITREX assumes this is a response and sends the frame out on a secondary address.</td>
</tr>
<tr>
<td>Example</td>
<td>This example sends a Receiver Not Ready supervisory command frame. The Subscriber would send this command on primary address 01. The Network would send it on primary address 03.</td>
</tr>
<tr>
<td></td>
<td>FPRNR</td>
</tr>
</tbody>
</table>

---

*TEKELEC*  
8.7-5  
6/11/90
### Numbered Responses

<table>
<thead>
<tr>
<th>Description</th>
<th>These commands send either a RNR, RR, or REJ response frame.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Statement or Direct</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td><strong>F(F)RNR(Nr)</strong> Sends a Receiver Not Ready frame.</td>
</tr>
<tr>
<td></td>
<td><strong>F(F)RR(Nr)</strong> Sends a Receiver Ready frame.</td>
</tr>
<tr>
<td></td>
<td><strong>F(F)REJ(Nr)</strong> Sends a Reject frame.</td>
</tr>
<tr>
<td></td>
<td><em>where: Nr is in the range 0 to 7 and F is an optional final bit.</em></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>This example sends a Receiver Ready indicating that the simulator has received a frame with Nr equal to 3. The Subscriber sends this response on secondary address 03, the Network on 01.</td>
</tr>
<tr>
<td></td>
<td><strong>FRR3</strong></td>
</tr>
</tbody>
</table>
Send Information Frame

Description
This command (FI) sends an I-frame. FI commands should be used only when the frame level has been established. The internal State variables are updated automatically; however, the values that are sent over the link may be overridden by specifying Ns and Nr.

Type
Statement or Direct

Syntax
\[ F(P)I(Ns),(Nr)(PACKET) \] Sends I-Frame with packet mnemonic. The I-Field is specified by the optional PACKET field. PACKET is defined using the Packet Level commands, which are described in the next section.

\[ F(P)I(Ns),(Nr)(PHh1h2...) \] Sends I-Frame with packet in hex.

\[ F(P)I(Ns),(Nr)(PAabcd...) \] Sends I-Frame with packet in ASCII.

Example 1
This example sends an unpolled I-Frame using the Packet Level command PRST (Restart packet) as the I-field. The automatic simulator provides the missing information.

FIPRST

Example 2
This example sends an unpolled user-defined I-Frame in hex.

FIPH010041424344

Example 3
This example sends a polled user-defined I-Frame in hex.

FIPPH010041424344
PACKET LEVEL COMMANDS

Introduction

Packet level commands allow you to transmit any type of X.25 packet. The Automatic X.25 Simulator is continually updating packet level counters and timers. However, as with the frame level commands, you can override this function with specific packet commands.

For example, the command PRST tells SITREX to send a packet level restart down to the link and wait for a restart confirmation.

Refer to Section 8.2 for a description of how packet level commands interact with the Automatic X.25 Simulator.
Send User-Defined Packet

Description
These commands send a user-defined packet in hex or ASCII. The link must be established at the frame level using frame level commands before these commands are used. These commands are useful for sending packets without specifying F(P)I (send I-frame) commands.

Type
Statement or Direct

Syntax

PPh.......h Sends packet in hex, where h....h is the contents of the packet, in hex.

PAa.......a Sends packet in ASCII, where a....a is the contents of the packet, in ASCII.

Example
This example sends a call to pseudo-user 5 in hex:

PH10010B071234505000
Send Call/Call Confirmation Packet

<table>
<thead>
<tr>
<th>Description</th>
<th>These commands send either a call or call confirmation packet. If a call is received and no sub-address has been defined (no pseudo-user called), the default sub-address will be pseudo-user 2 (local keyboard and screen).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>PUnCALL(D)(Na or V)(Nb)(Fh........h)(DHh........h) Sends a Call packet with the call data in hex.</td>
</tr>
<tr>
<td></td>
<td>PUnCALL(D)(Na or V)(Nb)(Fh........h)(DAa........a) Sends a Call packet with the call data in ASCII.</td>
</tr>
<tr>
<td></td>
<td>PUnCCALL(D) Sends a Call Confirmation packet.</td>
</tr>
</tbody>
</table>

n is the pseudo-user sending the call, in the range 1 - 7.
D is an optional delivery confirmation bit.
Na is the called address in decimal (15 digits maximum). The called address is the destination.
V is the called number entered when initially configuring the SITREX Menu.
Nb is the calling number in decimal (15 digits maximum).
F is the called facilities followed by an even number of bytes (64 maximum).
DH or DA is the call data in hex or ASCII (128 bytes maximum).

Example 1
This example sends a call packet to address 12345, sub-number 03 from the Chameleon's keyboard and screen (CCITT version). U2 is the pseudo-user number of the Chameleon keyboard.

PU2CALL1234503

Example 2
This example calls 12345, sub-number 03 from the Chameleon keyboard. The sub-unit address 03 is put into the 5th byte of data sent with the call request packet as specified by DATAPAC. "D" specifies data, "H" Hexadecimal.

PU2CALL12345,DH00 00 00 00 03
Send Supervisory Packet

Description
These commands send a Receiver Ready (RR), Receiver Not Ready (RNR) or Reject (REJ) packet. The internal state variables are updated automatically; however, the values that are sent over the link may be overridden by specifying Pr.

The packet is sent if the current status of the unit allows it to do so, unless:

- A call has not been established.
- A call is in the process of being established.
- A call is in the process of being cleared.

Type
Statement or Direct

Syntax
PUnRR(Pr) Sends a Receiver Ready packet.
PUnRNR(Pr) Sends a Receiver Not Ready packet.
PUnREJ(Pr) Sends a Reject packet.

n is the pseudo-user unit which is sending the packet, in the range 1 - 7.

For Modulo 8, Pr is in the range of 0 to 7. For Modulo 128, Pr is in the range of 000 to 127.

Example 1
This example sends an RR from pseudo-user unit 1 with Pr set to 0, in Modulo 8.

PU1RR0

Example 2
This example sends an RR from pseudo-user unit 1 with Pr set to 000, in Modulo 128.

PU1RR000
## Send Restart/Restart Confirmation Packet

**Description**  
These commands send a Restart or Restart Confirmation packet. These packets are always sent on logical channel 0.

**Type**  
Statement or Direct

**Syntax**  
`PRST(h1h2)( h3h4)` Sends a Restart packet, where `h1h2` specifies the cause code and `h3h4` specifies the diagnostic code. Clears every pseudo-user state.

`PCRST` Sends a Restart Confirmation packet.

**Example**  
This example sends a Restart packet with restart cause code 00.

`PRST00`
Send Clear/Reset/Interrupt Confirmation Packets

Description
These commands send Clear, Clear Confirmation, Reset, Reset Confirmation, Interrupt and Interrupt Confirmation packets. You must be in X.25 flow control ready state to send CLEAR commands.

Type
Statement or Direct

Syntax

**PUnCLEAR(h1h2)(h3h4)(,DHh...h)** Sends a Clear packet with data in hex.

**PUnCLEAR(h1h2)(h3h4)(,DAa...a)** Sends a Clear packet with data in ASCII.

**PUnCCLEAR** Sends a Clear Confirmation packet.

**PUnRSET(h1h2)(h3h4)** Sends a Reset packet.

**PUnCRSET** Sends a Reset Confirmation packet.

**PUnINT(h1h2)(h3h4)** Sends an Interrupt packet.

**PUnCINT** Sends an Interrupt Confirmation packet.

where: **D** is an optional delivery confirmation bit, **h1h2** specifies the cause code, and **h3h4** specifies the diagnostic code. Although two bytes can be entered, normally a Subscriber sends only the cause code byte, while a Network sends both the cause code and diagnostic code bytes.

In a Clear packet, data can be sent in a combination of hex and ASCII. Press the Escape key to switch between hex and ASCII data, as shown in the example below. A tilde (') indicates where the Escape key was pressed.

Example 1
This example sends a Call Confirmation Packet from pseudo-user unit 1.

**PU1CCALL**

Example 2
This example sends a CLEAR packet from pseudo-user unit 1 containing a combination of hex and ASCII data.

**PU1CLEAR01 02,DA DATA IN A CLEAR PACKET - OD 0A**
Send Data Packs

Description: These commands send a data packet with the data in hex or ASCII.

Type: Statement or Direct

Syntax:

PU\text{UnD}(Ps)(Pr)(Q)(M)(D)Hh\ldots h \quad \text{Sends a data packet with the data in hex.}

PU\text{UnD}(Ps)(Pr)(Q)(M)(D)Aa\ldots a \quad \text{Sends a data packet with the data in ASCII.}

n is the pseudo-user number, in the range 1 - 7.

Q is the optional Qualifier bit.

M is the optional More Data bit.

D is the optional Delivery confirmation bit.

Example 1: This example sends a data packet containing hex 41 42 43 on the logical channel associated with pseudo-user unit 1.

\text{PU1DH414243}

Example 2: This example sends a data packet with the Q, M and D bits, and five bytes of ASCII characters (HELLO) set on the logical channel associated with pseudo-user unit 4.

\text{PU4DQMDAHELLO}
Send Diagnostic Packet

Description
This command sends a diagnostic packet on logical channel 0.

Type
Statement or Direct

Syntax
PDIAGh1h2h3h4h5h6h7h8

where: h1h2 is the diagnostic byte, and h3h4h5h6h7h8 are the first three bytes of the header information from the packet (GFI, LCGN, LCN, Packet Type Identifier), in hex.

Example
In this example, aa is the diagnostic byte, and bb, cc, dd are the first three bytes of the header information from the erroneous packet.

PDIAGaabbccdd
PARAMETER COMMANDS

Introduction

Parameter (set) commands enable you to define the value of X.25 global parameters, such as frame window size. They also allow you to control the Automatic X.25 Simulator.

For example, the command SK3 sets the maximum frame window size to 3.

Refer to Section 8.2 for a description of how the command language interacts with the Automatic X.25 Simulator.
Set Frame Level

Description
The SFOF and SFON commands set the frame level off or on, respectively. If the Chameleon is configured in Initiative, or Stand-by mode, the default is frame level ON. If configured in Manual mode, the default will be frame level OFF.

Type
Statement or Direct

Syntax

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFON</td>
<td>Sets frame level ON.</td>
</tr>
<tr>
<td>SFOF</td>
<td>Sets frame level OFF.</td>
</tr>
</tbody>
</table>

The SFOF (frame level off) command puts the Simulator into Frame Manual (FM) mode, which is equivalent to Manual mode. It forces the Simulator to ignore the frames received, while maintaining automatic frame level mechanisms. Only SITREX commands will be executed.

In FM mode, outgoing frames are still processed by the automatic Simulator, and State variables relevant to outgoing frames are updated. The frame level State variables for received frames remain unchanged. Incoming frames can be processed by SITREX using WAIT commands.

If SITREX is in any frame level state other than Disconnect (without any outstanding frame or packet), and you turn the frame level OFF, the automatic Simulator will stay in the same state of transmission, taking actions relevant to that state.

The SFON (frame level on) command reactivates the processing of incoming frames. Outgoing frames are still processed by the automatic Simulator.

If numbered frames and packets are received in Frame Manual (FM) mode, SFON will reactivate the frame level Simulator with outdated State variables. These variables can be updated in two ways:

1. With a SITREX SET command, for example SNR+ or SNR-, increment or decrement Ns, or SUn PRx, set Pr of Pseudo-User n to value x.
2. By allowing the Simulator to restart with wrong State variables, thus forcing the other device to initiate recovery procedures.

Note
If you enter the **SLON** (Set Link ON) command, an implicit **SFON** will be executed.

**Example 1**
This example sets frame level on.

**SFON**

**Example 2**
This example sets frame level off.

**SF0F**
Set Packet Level

Description
The SPOF and SPON commands set the packet level off and on, respectively. If the Chameleon is configured in Initiative or Stand-by mode, the default is packet level ON. If configured in Manual mode, the default is packet level OFF.

Type
Statement or Direct

Syntax
SPON  Set packet level ON.
SPOF  Set packet level OFF.

SPOF forces the Simulator to ignore the packets received, while maintaining automatic frame level mechanisms. In this mode (Frame Automatic - Packet Manual = FA-PM), the frame level State variables are automatically updated.

The packet level State variable for received packets remain unchanged, but the packet level State variables for transmitted packets are updated.

SPOF does not disable the packet level completely, only the automatic processing of incoming packets is suppressed. Incoming packets can be processed by SITREX with WAIT commands.

SPON will reactivate the Simulator’s packet level. The frame level must be set ON before this command will take effect.

Example 1
This example sets packet level on.
SPON

Example 2
This example sets packet level off.
SPOF
Set Link Level

Description: The SLOF and SLON commands set the link level off or on, respectively.

Type: Statement or Direct

Syntax:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLON</td>
<td>Sets Link ON.</td>
</tr>
<tr>
<td>SLOF</td>
<td>Sets Link OFF.</td>
</tr>
</tbody>
</table>

**SLON** sends a SARM or a SABM to establish the link. The system waits for a UA until T1 expires, then sends polled SARMs or SABMs N2-1 times. If a UA is still not received, one DISConnect and then N2-1 number of polled DISConnects will be sent.

**SLON** must be used in program mode to establish a link before using any of the SITREX packet level commands. **SLON** activates the automatic frame level (SFON), which can be disabled using SFOF.

**SLOF** attempts to disconnect the line. Upon reception of a UA, the automatic Simulator enters the disconnected phase. (Refer to CCITT Red Book, Recommendation X.25, Section on Link Layer, paragraph on Link Set-ups: 2.4.5.3 and 2.4.5.4, for more information.)

**Example 1**

This example sets the link off.

SLOF

**Example 2**

This example sets the link on.

SLON
Force Link On

**Description**  The LNKUP command forces the Simulator to assume that the link has already been established. It assumes that a SABM has been sent and a UA has been received in response.

However, unlike the SLON command above, LNKUP does not actually send or receive the appropriate frames. Also, LNKUP activate the automatic frame level (SFON), which can be disabled using SFOF

Note: After the first transmission, if an answer is not received, SITREX will attempt to set the link on.

**Type**  Statement or Direct

**Syntax**  LNKUP

**Example**  This example forces the Simulator to assume that the link is up.

LNKUP
### Transmit CRC

**Description**
The SCRC command enables you to specify if subsequent frames should be sent with a good CRC or without a CRC. The default condition is that frames will be sent with a good CRC.

If the Simulator is sending frames without the CRC calculation, the last bytes are considered to be the CRC.

Instead of transmitting a frame with a calculated CRC, you may use the memory buffer to store the CRC at the final position, and later transmit the entire buffer. Basically, this feature allows you to build your own CRC.

The operations on variables allow you to calculate the CRC. It can be stored at the end of the message buffer using the commands SXAlhh and SXAIXB (see example below).

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>SCRC+</td>
<td>Specifies that subsequent frames include a good CRC.</td>
</tr>
<tr>
<td>Direct</td>
<td>SCRC-</td>
<td>Specifies that subsequent frames will be sent without a CRC.</td>
</tr>
</tbody>
</table>

**Example**
This example transmits the frame 01 02 03 04 with the CRC 39 91, where variable A = 39 and variable B = 91.

```
SXAH39:SXBH91 Variable A = 39 (MSB) of CRC, B to (LSB) of CRC
SCRC-    Specifies to send a frames without a CRC.
SXCH02   Initializes C to equal CRC length.
SMH01020304 Stores the message to be transmitted.
SXL000   Extracts the length of the message buffer (byte 00) and put into variable L.
SXL+XC   Adjust the length of the buffer.
SXL100   Insert the new length into byte 0 of the buffer.
SXIHO1   Assigns variable I to 1.
SXBIXL   Assigns variable B to last position of the buffer.
SXL-XI   Decrements the length in variable L.
SXAIXL   Assigns variable A to position before the last in the buffer.
FM       Transmits the frame.
SCRC+    Resumes sending frames with CRC.
```
### Set Primary/Secondary Address

<table>
<thead>
<tr>
<th>Description</th>
<th>The SPA and SSA commands set the primary and secondary addresses, respectively.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>SPAh1h2 Sets the Primary Address.</td>
</tr>
<tr>
<td></td>
<td>SSAh1h2 Sets the Secondary Address.</td>
</tr>
<tr>
<td></td>
<td>where: h1h2 is the address. If h1 is absent, it is set to 0.</td>
</tr>
<tr>
<td>Example</td>
<td>This example sets the primary address to 03.</td>
</tr>
<tr>
<td></td>
<td>SPA03</td>
</tr>
</tbody>
</table>
Set Frame and Packet Window Size

**Description**
These commands set the frame window and packet window sizes.

**Type**
Statement

**Syntax**

- **SKx**
  Sets the frame window size, where x is in the range 1 - 3.

- **SUnW(R or T)x**
  Sets the packet window size, where x is the window size in the range 1 - 7.
  T specifies the Transmit window size.
  R specifies the Receive window size.
  n is the pseudo-user number in the range 1 - 7.

**Example 1**
This example sets the packet window size to 7 for pseudo-user unit 1.

SUnW7

**Example 2**
This example sets the frame window size to 2.

SK2
Set Data Packet Length

Description
These commands set the maximum length of the transmitted or received data packet for the logical channel. These values must be (re)set before call set-up and transmission. These parameters can be modified only by clearing the channel and returning to a ready state.

The packet length is recognized only if the initial length specified for the Frame Size (N1) parameter in the Frame & Packet Level Parameter Set-up Menu is sufficient.

Type
Statement

Syntax
SUnLTnnn 
Sets the maximum length of the transmitted data packet.

SUnLRnnn 
Sets the maximum length of the received data packet.

SGTh1h2 
Sets the length of the data in the data packet sent by a traffic generator, where h1h2 can be any value between 00 and F8. (Since the frames have a maximum length of 255 bytes, h1h2 cannot equal a value from F9 to FF.)

where: n is the pseudo-user number in the range 1 - 7 and nnn is one of the packet lengths supported by SITREX: 016, 032, 064, and 128.

Example
This example sets the maximum length of a data packet transmitted from pseudo-user unit 2 to 64 bytes.

SU2LT064
Set Frame and Packet State Variables

Description

These commands increment, decrement, or set the values of N(s), N(r), P(s) and P(r). To use the commands which affect modulo 8 or modulo 128, you must initialize the SITREX User Facilities Menu. (See Section 8.3 for information.) If the sequence numbering method chosen at the packet level is modulo 128, all the commands must be for modulo 128.

Type

Statement or Direct

Syntax

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNS</td>
<td>Increments N(s).</td>
</tr>
<tr>
<td>SNS-</td>
<td>Decrements N(s).</td>
</tr>
<tr>
<td>SNSx</td>
<td>Sets N(s) to a value between 0 and 7.</td>
</tr>
<tr>
<td>SNR+</td>
<td>Increments N(r).</td>
</tr>
<tr>
<td>SNR-</td>
<td>Decrements N(r).</td>
</tr>
<tr>
<td>SNRx</td>
<td>Sets N(r) to a value between 0 and 7.</td>
</tr>
<tr>
<td>SUnPR+</td>
<td>Increments P(r).</td>
</tr>
<tr>
<td>SUnPR-</td>
<td>Decrements P(r).</td>
</tr>
<tr>
<td>SUnPRx(xx)</td>
<td>Sets value of P(r). The range of x is 0 - 7 for</td>
</tr>
<tr>
<td></td>
<td>Mod 8 and 000 - 127 for Mod 128. n is the</td>
</tr>
<tr>
<td></td>
<td>pseudo-user number.</td>
</tr>
<tr>
<td>SUnPS+</td>
<td>Increments P(s).</td>
</tr>
<tr>
<td>SUnPS-</td>
<td>Decrements P(s).</td>
</tr>
<tr>
<td>SUnPSx(xx)</td>
<td>Sets the value of P(s). The range of x is 0 - 7 for</td>
</tr>
<tr>
<td></td>
<td>Mod 8 and 000 - 127 for Mod 128. n is the</td>
</tr>
<tr>
<td></td>
<td>pseudo-user number.</td>
</tr>
</tbody>
</table>

Example 1

This example decrements N(s).

SNS-

Example 2

This example sets the value of P(s) using Mod 128 at 000.

SU3PS000
Set Logical Channel Group Number

Description: The SLG command assigns a Logical Channel Group Number (LCGN) as a default value to the next placed call.

Type: Statement or Direct

Syntax: SLGh1

where: \( h1 \) is the default LCGN in hex, in the range 0 to F.

Example: This example assigns hex 5 as the default Logical Channel Group Number.

SLG5
Set Interface Leads

Description  This command sets a specific interface lead to logical 1 or 0.

Type  Statement or Direct

Syntax  

| Snnn+ | Sets interface lead nnn active (space). |
|------------------|
| Snnn- | Sets interface lead nnn inactive (mark). |

where: nnn is one of the signal numbers in the chart below.

<table>
<thead>
<tr>
<th>Chameleon</th>
<th>INTERFACE LEADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Simulating a DCE</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>CTS</td>
</tr>
<tr>
<td>107</td>
<td>DSR</td>
</tr>
<tr>
<td>109</td>
<td>DCD</td>
</tr>
<tr>
<td>122</td>
<td>SDCD</td>
</tr>
<tr>
<td>125</td>
<td>RI</td>
</tr>
<tr>
<td>Simulating a DTE</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>RTS</td>
</tr>
<tr>
<td>108</td>
<td>DTR</td>
</tr>
</tbody>
</table>

Example  This example sets the DCD (signal 109) to space.

S109+
Test Interface Leads

Description
This command tests an interface lead for logical 1 or 0. If the test is true, a specified program line is executed. During the execution of a scenario, this command permits different actions depending on the status of the interface signal. All the signals in the table below can be tested for both the DTE and DCE.

Type
Statement or Direct

Syntax

- **IF**nnn + dddd: If interface signal is active, goes to line number **ddd**. nnn is a signal number in the table below.
- **IF**nnn- dddd: If interface signal nnn is inactive, goes to line number **ddd**. nnn is a signal number in the table below.

<table>
<thead>
<tr>
<th>Chameleon</th>
<th>INTERFACE LEADS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SIGNAL</td>
</tr>
<tr>
<td>Simulating a DCE</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>CTS</td>
</tr>
<tr>
<td>107</td>
<td>DSR</td>
</tr>
<tr>
<td>109</td>
<td>DCD</td>
</tr>
<tr>
<td>122</td>
<td>SDCD</td>
</tr>
<tr>
<td>125</td>
<td>RI</td>
</tr>
<tr>
<td>Simulating a DTE</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>RTS</td>
</tr>
<tr>
<td>108</td>
<td>DTR</td>
</tr>
</tbody>
</table>

Example

- **10 IF 106+40** If interface lead 106 is active, the scenario branches to line number 40.
- **20 PRINT 106** Displays status of CTS (106) if inactive.
- **30 GOTO 50** If inactive, branches to line 50.
- **40 PRINT 106** Displays status of CTS (106), if active.
- **50 REM STOP** Stops program execution.
Set Timers

<table>
<thead>
<tr>
<th>Description</th>
<th>These commands enable you to set the value of various SITREX timers. All of the SITREX timers count down to a terminal value of 0. Timer T' and T&quot; are used internally by SITRES and are initialized automatically.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement</td>
</tr>
<tr>
<td>Syntax</td>
<td>ST'h1h2h3h4 Sets timer T', the timer associated with sending command frames (not I-Frames). It counts in tens of milliseconds. It is used for command re-transmission only. T' does not change the value of T1.</td>
</tr>
<tr>
<td></td>
<td>ST&quot;h1h2 Sets timer T&quot;, the timer associated with sending I-Frames from the Simulator. It is used for pacing transmission of I-Frames. It counts in units of seconds.</td>
</tr>
<tr>
<td></td>
<td>SWTh1h2h3h4 Sets the Watch-dog Timer, which is associated with the WAIT commands. It counts in tens of milliseconds.</td>
</tr>
<tr>
<td></td>
<td>STUh1h2h3h4 Sets timer TU, a user-defined timer. Timer TU is set to h1h2h3h4 * 10 milliseconds and counts down in tens of milliseconds.</td>
</tr>
<tr>
<td>Example</td>
<td>ST'1234</td>
</tr>
</tbody>
</table>

h1h2h3h4 indicates that four hex numbers must be entered as the timer value, in tens of milliseconds. h1h2 indicates that two hex numbers must be entered as the timer value in units of seconds.
### Set Counters

**Description**  
The SC command increments, decrements, or sets the values of a user-defined counter. You can use up to 8 counters. Refer to the IFC command to test the value of counters.

**Type**  
Statement

**Syntax**  
- **SCnh1h2**  
  Sets counter where, \( n \) is the counter number in the range 0 - 7 and \( h1h2 \) is the value in hex.
- **SCn+**  
  Increments counter, where \( n \) is the counter number in the range 0 - 7.
- **SCn-**  
  Decrements counter, where \( n \) is the counter number in the range 0 - 7.

**Example**  
This example decrements counter 2.  

```
SC2-
```
Set Pseudo-User Type

Description
This command sets or alters the attributes of a pseudo-user. SITREX supports seven pseudo-users, but two of these are internally reserved and cannot be altered using this command.

This command also associates a particular pseudo-user with a Logical Channel Group Number (LCGN) and a Logical Channel Number (LCN).

The type of a pseudo-user can be changed at any time. For example, a pseudo-user can be changed after a call has been placed, or after several data packets have been transferred. The default settings for the 7 pseudo-users are:

<table>
<thead>
<tr>
<th>Pseudo-User</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First Local Screen/Keyboard (Reserved for Trace page)</td>
</tr>
<tr>
<td>2</td>
<td>Second Local Screen/Keyboard (Reserved for Simulation page)</td>
</tr>
<tr>
<td>3</td>
<td>Traffic Generator (continuously transmits characters)</td>
</tr>
<tr>
<td>4</td>
<td>Echo Generator (returns received packets on same LCN)</td>
</tr>
<tr>
<td>5</td>
<td>Data Absorber (normal terminal)</td>
</tr>
<tr>
<td>6</td>
<td>Traffic Generator</td>
</tr>
<tr>
<td>7</td>
<td>Traffic Generator</td>
</tr>
</tbody>
</table>

Type

Statement or Direct

SPUnA  Defines the pseudo-user as a Data Absorber (a normal terminal).

SPUnE  Defines the pseudo-user as an Echo Generator, which returns received data packets on the same logical channel.

SPUnT  Defines the pseudo-user as a Traffic Generator, which continuously transmits characters.

where: n is the pseudo-user number in the range of 3 to 7, identified by the sub-number or port identification of the X.25 address. Pseudo-users 1 and 2 cannot be changed.

Example

This example defines pseudo-user 4 as a traffic generator.

SPU4T
Set Up PVCS and SVCS

Description
These commands set up a Permanent Virtual Circuit (PVC) or a Switched Virtual Circuit (SVC).

You must set PVC's before the link is established. A PVC is removed by sending or receiving a CLEAR. This enables you to reassign a PVC.

Type
Statement or Direct

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUnVPh1h2h3</td>
<td>Sets up a Permanent Virtual Circuit (PVC).</td>
</tr>
<tr>
<td>SUnVCh1h2h3</td>
<td>Sets up a Switched Virtual Circuit (SVC). It also associates the Logical Channel with the pseudo-user for the next call set-up.</td>
</tr>
</tbody>
</table>

where: \( n \) is the physical unit number, \( h1 \) the Logical Channel Group Number, and \( h2h3 \) is the Logical Channel Number.

Example
This example sets up a PVC with pseudo-user 5:

SU5VP001
WAIT COMMANDS

Introduction

The WAIT commands analyze the next item received and compare the incoming data to a frame, packet, or byte mask configuration specified in the command.

The WAIT commands enable you to do the following:

- Wait for a specific frame or packet command before the scenario proceeds
- Wait for a specific bit configuration before the scenario proceeds
- Wait for a specific frame or packet command and store the item in the message buffer
- Wait for a specific bit configuration and store the item in the message buffer

If the trace buffer has been switched OFF (STOF), a WAIT command will start the trace capture again. Only one WAIT command can be used at a time. For multiple triggering, refer to the WS and WST (WAIT and STORE) commands.

WAIT Timers

The WAIT commands depend on the value of a watch-dog timer, the address of the watch-dog timer, and a jump address. The WAIT command should not be executed until these three values are assigned, using the following commands:

- Set Watch-Dog Timer using the SWT command. If the timer expires before a frame or packet is received, it causes the scenario to jump to the Watch-Dog address set with the SADRWT command.

- Set Watch-Dog address using the SADRWT command. If no frame is received within the time set by the Watch-dog Timer, execution continues at the line number in the SADRWT command. If there is no SADRWT, the Watch-dog Timer will be inactive. Therefore, if no frame is received within the required time, an unpredictable error will result.

- Set the jump address using the SELSE command. SELSE specifies which line number to execute if something other than the item specified in the WAIT command is received. If a SELSE command is not executed before a WAIT command, an error results.
The **WAIT** commands are executed as follows:

Has a frame been received within the watch-dog timer period?

- Yes
  - Continue Scenario
- No
  - Jump to SADRWT address

Is it the frame expected?

- Yes
  - Continue Scenario
- No
  - Jump to SELSE address
## WAIT FOR FRAME OR PACKET

### Description
These commands cause the scenario to wait for the reception of a specific type of frame or packet. If the received item matches the item specified in the WAIT command, either the scenario continues, or the item is stored in the message buffer.

If no item is received within the time specified in the SWT command, the line specified in the SADRWT command is executed. If something other than the specified item is received, the line specified in the SELSE command is executed.

### Type
Statement

### Syntax

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF(command)</td>
<td>Waits for a specified type of frame before executing next line of scenario. command is any SITREX Frame Level command.</td>
</tr>
<tr>
<td>WSF(command)</td>
<td>Waits for a specified type of frame and stores it in the message buffer. command is any SITREX Frame Level command.</td>
</tr>
<tr>
<td>WP(command)</td>
<td>Waits for a specified type of packet before executing next line of scenario. command is any SITREX Packet Level command except a Call or Call Confirmation.</td>
</tr>
<tr>
<td>WSP(command)</td>
<td>Waits for a specified type of packet and stores it in the message buffer. command is any SITREX Packet Level command except a Call or Call Confirmation.</td>
</tr>
</tbody>
</table>

You cannot wait for a Call, Call Confirmation, or any frame or packet from pseudo-user 7. However, you can synchronize on these packets using the following syntax:

```
WPHh1h2h3h4 h......h
```

which causes the scenario to wait for a user-defined packet in hex.
Example 1  
This example waits for a user-defined frame in hex, in this case, a SABM on address 03.

WFH032F

Example 2  
This example waits for a UA response frame with the final bit set to 1.

WFFUA

Example 3  
This example waits for a RR3 packet sent to pseudo-user 1.

WPU1  RR3
WAIT FOR BYTE MASK CONFIGURATION

Description
This command causes the Chameleon to wait for a specific byte mask configuration. A byte mask configuration is a string of bytes associated with 8-bit masks.

The WT command is a more general syntax than WP or WF. Its use is recommended when waiting for frames and packets, such as RRn, RNRn, where n is not known. When waiting for these frames, the command WFRR will wait for the currently expected Nr.

The Chameleon remains idle until a frame is received. If the frame matches the configuration in the WAIT command, either the next line is executed (WT) or the item is stored in the message buffer (WST).

If something other than the specified item is received, the line specified in the SELSE command is executed.

Type
Statement

Syntax

WTXX/YY(„XX/YY)(...) Waits for specified byte mask and executes next line of scenario when received.

WSTXX/YY(„XX/YY)(...) Waits for a byte mask and stores it in the message buffer when received.

The first XX/YY pair corresponds to the first byte in the received frame, the second pair corresponds to the second byte, and so on. An XX/YY pair is separated with a slash(/). Two successive XX/YY pairs are separated by a comma (,).

XX is two hex characters that define the expected binary configuration after masking. YY is the 8-bit mask. Use X to indicate that you do not care what is received.

YY is logically ANDed with its corresponding byte position in the receive buffer. The result is compared to XX. An exact match indicates that the condition is true, and the next comparison is performed.

You cannot use the WP (wait for packet) or WF (wait for frame) commands to wait for a Call, Call Confirmation packet, or any frame or packet from pseudo-user 7.

You can use the WT command to synchronize on these, using the following syntax:

WT 00/00,00/00,00/00/,00/00,0B/FF
Example 1

This example waits for a polled or unpoll SABM on any address.

WT00/00,2F/EF

Example 2

The following example shows the mask configuration used when waiting for a polled or unpoll SABM on any address.

10 SWT1000
20 SADRWT 200
30 SELSE 40
40 WST00/00,00/00,00/00,00,00,0B/FF
50 IS00/00,00/00,00/00,07/FF 100
60 PRINT THE LOGICAL CHANNEL NUMBER IS NOT 7
70 GOTO 210
100 PRINT THE LOGICAL CHANNEL NUMBER IS CORRECT
200 PRINT TIMER EXPIRED
210 REM END OF SCENARIO
1000 REM STOP

XXXXXXX 001X1111 Expected Result in Binary (SABM).

X indicates that you do not care what is received. The X in the second byte is the Poll/Final bit.

00000000 00101111 Expected Result in Binary.
00 2F Expected Result in HEX.
00000000 11101111 Mask in Binary.
00 EF Mask in HEX.

WT00/00,2F/EF SITREX command used to specify this particular byte mask.

00000001 00111111 Incoming Data -- Polled SABM on address 01.

The Mask is ANDed with the incoming data and the intermediate result is:

00000000 00101111 In Binary.
00 2F In HEX.

This is compared with the expected result shown above. The condition specified by the WAIT command is evaluated as TRUE.
When writing scenarios, remember that SITREX commands take time to execute. Therefore, if a frame is received while SITREX is setting up its internal parameters, that frame may not be caught by SITREX. For example:

5  SFOF
6  FSABM
10 SELSE 20
20 WFH0363
30 FH0363
40 SELSE 50
50 WFH0363

While SITREX is executing line 10 or line 40, an incoming frame will not be caught because SITREX is setting the wait address. The same scenario would be better written as:

5  SFOF
10 SELSE 20
15 FSABM
20 WFH0363
35 FDISC
30 SELSE 50
40 FH0363
50 WFH0363

The SELSE command will be completed before the next frame is transmitted, and a frame received during the execution of line 50 will be caught.
### SET WAIT JUMP ADDRESSES

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>These commands set the addresses that are used in conjunction with all WAIT commands. SADRWT specifies the line number to execute if a frame is not received within the time specified by the Watch-Dog Timer (SWT). SELSE specifies the line number to execute if something is received, but it is not the item specified in the WAIT command.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SADRWT dddd</td>
<td>Sets the jump address for the Watch-Dog Timer. Causes the scenario to jump to line dddd if the Watch-dog Timer elapses before an item is received from the line.</td>
</tr>
<tr>
<td>SELSE dddd</td>
<td>Causes the scenario to jump to line dddd if the next received item is not the one specified in the WAIT command.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 SWT1000</td>
</tr>
<tr>
<td>20 SADRWT 90</td>
</tr>
<tr>
<td>30 SELSE 70</td>
</tr>
<tr>
<td>40 WFSARM</td>
</tr>
<tr>
<td>50 PRINT SARM Rec'd</td>
</tr>
<tr>
<td>60 GOTO 100</td>
</tr>
<tr>
<td>70 PRINT NOT A SARM</td>
</tr>
<tr>
<td>80 GOTO 100</td>
</tr>
<tr>
<td>90 PRINT TIMER ELAPSED</td>
</tr>
<tr>
<td>100 REM SCENARIO END</td>
</tr>
</tbody>
</table>
SET WATCHDOG TIMER

Description
The SWT command sets the length of the Watch Dog Timer for WAIT commands. It must be set before any of the WAIT commands are executed. If the Watch Dog Timer expires before an item is received from the line, the scenario executes the line number specified in the SADRWT command.

Type
Statement

Syntax
SWTxxxx
where: xxx is in units of tens of milliseconds (.0001)

Example
SWT1000
Sets the Watchdog Timer to wait for approximately 4 seconds. Note that the SADRWT command must be used to specify the line number to jump to when this timer expires.
**LOOP, JUMP, AND REINITIALIZATION COMMANDS**

**Introduction**

This section contains commands that do the following:

- Set up loops to execute a list of commands a specific number of times
- Conditional jump commands (IF commands) that enables you to jump to a specified line number based on the value of a timer or counter
- Unconditional jump command (GOTO, GOSUB)
- Reinitialization commands that enable you to:
  - Stop program execution
  - Exit from Sitrex command mode and return to the main menu
  - Exit from Sitrex command mode and return to the Automatic X.25 Simulator
SET UP PROGRAM LOOP

Description
These commands set up a loop that executes a list of commands a specified number of times. Nested loops are not allowed.

Type
Statement

Syntax
\[ *h1h2 \]
Signals the beginning of the loop, where \( h1h2 \) specifies the number of times to execute the following commands, within the range 1 to FF.

\[ ; \]
Signals the end of the loop. ; must occur on a line by itself.

The symbols * and ; must each appear at the beginning of the line. The program lines between * and ; are repeated \( h1h2 \) times. When the loop is completed, the first command following ; is executed.

Example
This example sets up a loop that sends four SARMs followed by one DISC.

<table>
<thead>
<tr>
<th>Line</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>*04</td>
<td>Sets loop to repeat 4 times.</td>
</tr>
<tr>
<td>20</td>
<td>FSARM</td>
<td>Send a SARM.</td>
</tr>
<tr>
<td>30</td>
<td>;</td>
<td>Ends loop segment.</td>
</tr>
<tr>
<td>40</td>
<td>FDISC</td>
<td>Send a DISC.</td>
</tr>
</tbody>
</table>
**CONDITIONAL JUMP (IF)**

**Description**
Tests a timer or counter for 0. If test is true, jumps to specified line number. Otherwise, the next command will be executed.

**Type**
Statement

**Syntax**
- **IFT' dddd** Tests timer T'.
- **IFT" dddd** Tests timer T".
- **IFTU dddd** Tests user-defined timer TU.
- **IFCn dddd** Tests counter, where n is the counter number in the range 0 - 7.

where: dddd is the line number to jump to, if the value of the timer or counter is zero.

**Example 1**
This example checks the user timer and prints a message when it expires.

```
10 STU0300  Sets user timer to 300(Hex) milliseconds.
20 IFTU 40   If timer TU equals 0, go to 40.
30 GOTO 20   If timer TU is not equal to 40, jumps to line 20 and tests again.
40 PRINT TIME UP  When timer expires, display message.
```
UNCONDITIONAL JUMP (GOTO, GOSUB)

Description
The GOTO and GOSUB commands enable you to unconditionally jump to a specified line number in a SITREX scenario. GOSUB jumps to a line number that executes a subroutine that ends with the RETURN command. When a RETURN is encountered, the program jumps to the line following the the GOSUB.

Type
Statement

Syntax
GOTO dddd Unconditionally jump to line number dddd. Do not use GOTO within a repeated loop.

GOSUB dddd Unconditionally jump to line number dddd, execute the subsequent lines as a subroutine. Return to the instruction following the GOSUB after encountering a RETURN.

RETURN Signals end of subroutine. Causes scenario to jump to the instruction following the GOSUB. If a RETURN is encountered without a preceding GOSUB, an error 18 will result.

Example
This example waits for frames and executes subroutines when received.

10 SELSE 20 If frame specified in WAIT command (line 20) is not received, jump to line 20.

20 WTOO/00 Waits for any incoming frame.

30 GOSUB 100 Executes a subroutine beginning at line 100.

40 SELSE 10 If frame specified in WAIT command (line 50) is not received, jump to line 10.

50 WFCMDR34 Waits for CMDR frame.

60 GOTO 10 Unconditionally jumps to line 10.

100 PRINT OK Subroutine. Print OK.

101 RETURN Returns from subroutine to.
## REINITIALIZATION

<table>
<thead>
<tr>
<th>Description</th>
<th>These three commands enable you to reinitialize SITREX by stopping program execution or exiting from SITREX command mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Direct</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td><strong>HALT</strong> Exits SITREX and returns to the Chameleon main menu.</td>
</tr>
<tr>
<td></td>
<td><strong>EXIT</strong> Exits SITREX command mode and returns to the SITREX Automatic X.25 Simulator.</td>
</tr>
<tr>
<td></td>
<td><strong>ESCape key</strong> Interrupts a scenario during execution. The system displays the following message: *** JOB CANCELLED ***</td>
</tr>
<tr>
<td><strong>Example 1</strong></td>
<td>This example exits command mode and displays the Chameleon main menu.</td>
</tr>
<tr>
<td></td>
<td><strong>HALT</strong></td>
</tr>
<tr>
<td><strong>Example 2</strong></td>
<td>This example exits SITREX command mode and displays the message <strong>SITEX IDLE</strong>.</td>
</tr>
<tr>
<td></td>
<td><strong>EXIT</strong></td>
</tr>
</tbody>
</table>
PROGRAM MANAGEMENT COMMANDS

Introduction

This section contains the command that enable you to develop your SITREX scenarios. Many of the commands in this section are direct commands. That is, they can be used only at the ! prompt and cannot be included in a scenario.

The commands in this section enable you to:

• Save scenarios to disk
• Load scenarios into memory
• Run programs
• Clear programming memory so that you can enter a new program
## CHAIN PROGRAMS

<table>
<thead>
<tr>
<th>Description</th>
<th>The &amp; command clears the current scenario from memory, loads a specified scenario into memory, and executes it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement</td>
</tr>
<tr>
<td>Syntax</td>
<td>&amp;xfilename</td>
</tr>
</tbody>
</table>

where: filename is the name of the scenario to be loaded and x is either 0 (hard drive) or 1 (floppy drive) and specifies the drive where the scenario is stored.

<table>
<thead>
<tr>
<th>Example</th>
<th>This example sends a SABM and waits for a UA. When the response is received, it clears the memory, and loads and executes the scenario TEST1510.</th>
</tr>
</thead>
</table>

```
10    SWTFFFFF       Set watch-dog timer to FFFF.
20    SADRWT 100     Set watchdog jump address to 1000.
30    SELSE 50        Set WAIT jump address to 50.
40    FSABM           Transmit a SABM.
50    WFUA            Wait for a UA.
60    PRINT TEST OK  Print TEST OK if received UA.
70    GOTO 200        Jump to 2000.
100   PRINT WT ELAPSED If UA is not received, and the timer elapses, print error message.
110   GOTO 300        Jumps to line 300 if watch-dog timer expires.
200   &OTEST1510      Loads and executes program file TEST1510 from the hard disk.
300   REM STOP        
```
LOAD PROGRAM

Description
This command loads a scenario file into memory.

Type
Direct

Syntax
LOAD

The system will respond with the prompt:

Filename:

Enter the filename using the format:

xfilename

where: filename is the name of the scenario to load and x is either 0 (hard drive) or 1 (floppy drive) and specifies the drive where the scenario is stored.

Example
This example loads the scenario named PROGRAM from the hard drive.

LOAD

Filename: 0PROGRAM
**REMARK**

<table>
<thead>
<tr>
<th>Description</th>
<th>The REM command enables you to enter programming remarks in a scenario file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement</td>
</tr>
<tr>
<td>Syntax</td>
<td>REM (text)</td>
</tr>
<tr>
<td>Example</td>
<td>10 REM Written by Tekelec</td>
</tr>
<tr>
<td></td>
<td>20 REM January 1988</td>
</tr>
<tr>
<td></td>
<td>30 REM Example remarks</td>
</tr>
</tbody>
</table>
### NEW PROGRAM

<table>
<thead>
<tr>
<th>Description</th>
<th>This command erases the scenario in memory so that a new program can be written.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>NEW</td>
</tr>
</tbody>
</table>
| Example     | !10 REM This is a remark.  
!LIST  
!LIST  
Result: 10 REM This is a remark.  
!NEW  
!LIST  
Result: ! |
### RUN PROGRAM

<table>
<thead>
<tr>
<th>Description</th>
<th>This command executes the scenario in memory. This is only applicable to a scenario with labeled commands which has been loaded from disk using the LOAD command, or entered through the keyboard. Pressing the ESCape key aborts scenario execution. While the scenario is running, you can view the trace by pressing T.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td>RUN</td>
</tr>
<tr>
<td>Example</td>
<td>This example executes the program currently in memory.</td>
</tr>
<tr>
<td></td>
<td>!LIST</td>
</tr>
<tr>
<td></td>
<td>!10 PRINT I'm a heavy SITREX user.</td>
</tr>
<tr>
<td></td>
<td>!RUN</td>
</tr>
<tr>
<td>Result:</td>
<td>I'm a heavy SITREX user.</td>
</tr>
<tr>
<td></td>
<td>!</td>
</tr>
</tbody>
</table>
### SAVE PROGRAM

**Description**
This command saves the scenario in memory to disk.

**Type**
Direct

**Syntax**
Save

The system responds with the prompt:

**Filename:**
Enter the filename using the format:

```
xfilename
```

where: `filename` is the name of the scenario to save and `x` is either 0 (hard drive) or 1 (floppy drive) and specifies the drive.

**Example**
This example saves the scenario in memory to a file named PROGRAM on the hard drive.

```
Save
Filename: 0PROGRAM
! 
```
NUMERIC VARIABLE COMMANDS

Introduction

This section contains the commands that enable you to define, manipulate, and test SITREX numeric variables.

You can define and store up to 26 numeric variables designated as XA - XZ and manipulate them as follows:

- Assign values directly or through user input
- Perform arithmetic operations
- Perform logical operations
- Shift and rotate the contents of a variable
- Test the contents of a variable
## VARIABLE OPERATIONS

**Description**

SITREX allows you to define and store up to 26 numeric variables (XA through XZ). The following commands enable you to assign values to variables, and perform arithmetic and logical operations using them.

Variables cannot contain hex values greater than FF. All calculations are Modulo 256 with the result of the operation stored in the leftmost variable.

**Type**

Statement

**Syntax**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXAHh1h2</td>
<td>Assigns a value to XA, where h1h2 is the value in hex.</td>
</tr>
<tr>
<td>SXA + XB</td>
<td>Adds XA and XB and stores the result in XA.</td>
</tr>
<tr>
<td>SXA - XB</td>
<td>Subtracts XB from XA and stores the result in XA.</td>
</tr>
<tr>
<td>SXA . XB</td>
<td>Logical AND between XA and XB.</td>
</tr>
<tr>
<td>SXA / XB</td>
<td>Logical OR between XA and XB.</td>
</tr>
<tr>
<td>SXA @ XB</td>
<td>Logical Exclusive OR (XOR) between XA and XB.</td>
</tr>
</tbody>
</table>

where: 
- S is the command
- XA and XB are variables
- H, +, - , ., /, and @ indicate the operation to be performed

**Example**

This example assigns values to two variables and then adds them together.

<table>
<thead>
<tr>
<th>Value</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SXAH23</td>
<td>Stores 23H in variable XA.</td>
</tr>
<tr>
<td>20</td>
<td>SXBH7B</td>
<td>Stores 7BH in variable XB.</td>
</tr>
<tr>
<td>30</td>
<td>SXA + XB</td>
<td>Adds variables AX and XB and stores result in XA.</td>
</tr>
</tbody>
</table>

Result

XA contains the value 9E hex.
SHIFT AND ROTATE VARIABLE CONTENTS

Description
These commands enable you to shift or rotate the contents of a numeric variable to the right or left. You can define up to 26 SITREX numeric variables using XA - XZ.

In shift commands, shifted bits fall outside the byte boundary and are discarded. This means that bits that are shifted will be 0.

In rotate commands, rotated bits are rotated at the other end of the byte. This causes bits that fall outside of one end to be rotated in at the other end.

Type
Statement

Syntax
SXADn  Shifts the contents of XA n times to the right.
SXAGn  Shifts the contents of XA n times to the left.
SXARn  Rotates the contents of XA n times to the right.
SXALn  Rotates the contents of XA n times to the left.

where: n is in the range 1 to 7.

Example
This example shifts XA three times left. XA contains the value 08H.

SXAG3

Result:  XA = 10H
### KEYBOARD INPUT

**Description**
The INPUT command causes a scenario to wait for the user to enter a two-digit hex value and then assigns the value to a numeric variable. If you are displaying the trace when an INPUT command is executed, it will pause until you enter a correct value.

Note: CTRL Z clears the screen.

**Type**
Statement

**Syntax**
```
INPUT XA
```

The following prompt appears:

```
XA = ?
```

The scenario will not continue until you enter two hex digits and the input value is stored in variable XA.

If you input an illegal response, the system will continue to ask for a new value until you enter a correct value, or terminate the scenario by pressing ESCape.

**Example**
This example prompts for a value for variable XA and then shifts the contents.

```
10 INPUT XA
20 SXAD2
```

Request user to enter value for XA.
Shift variable XA two places right.
## TEST VARIABLES

<table>
<thead>
<tr>
<th>Description</th>
<th>These commands test numeric variables against other variables or specified values and causes the scenario to jump to a specified line number if the test result is true.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Statement</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
</tr>
</tbody>
</table>

**IXA=XB dddd**  
If XA equals XB, go to line dddd.

**IXA=h1h2 dddd**  
If XA equals h1h2, go to line dddd.

**IXA#XB dddd**  
If XA is not equal to XB, go to line dddd.

**IXA#h1h2 dddd**  
If XA is not equal to h1h2, go to line dddd.

**IXA<XB dddd**  
If XA is less than XB, go to line dddd.

**IXA<h1h2 dddd**  
If XA is less than h1h2, go to line dddd.

**IXA>XB dddd**  
If XA is greater than XB, go to line dddd.

**IXA>h1h2 dddd**  
If XA is greater than h1h2, go to line dddd.

**IXA(XB dddd**  
If XA is less than or equal to XB, go to line dddd.

**IXA(h1h2 dddd**  
If XA is less than or equal to h1h2, go to line dddd.

**IXA)XB dddd**  
If XA is greater than or equal to XB, go to line dddd.

**IXA)h1h2 dddd**  
If XA is greater than or equal to h1h2, go to line dddd.

**Example**  
This example tests whether variables XA and XB are equal. If so, it prints OK!; if not, it prints ERROR!.

```
10  IXA=XB 40  
20  PRINT ERROR!
30  GOTO 50
40  PRINT OK!
50  REM END OF TEST
```
MESSAGE BUFFER COMMANDS

DEFINE MESSAGE FOR MESSAGE BUFFER

Description
These commands modify the contents of the message buffer. The message buffer can contain up to 64 bytes, numbered 1 through 40 hexadecimal. Byte 0 contains the length of the message in the buffer. This is illustrated in the figure below.

If you assign the contents of the buffer with the commands SMH or SMA, the system assigns the length (byte 0) automatically. If you assign data that exceeds 64 (40 hex) bytes, only the first 64 bytes will be accepted.

The DISPM command allows you to see the contents and the length of the message buffer.

Type

Statement

SMHh...h  Writes hexadecimal values into the buffer, where h...h is a maximum of 128 hex digits.

SMAa...a  Writes the ASCII characters into the buffer, where a...a is a maximum of 64 ASCII bytes.

SXAIXB  Inserts the value of variable XA in the byte of the buffer indicated by the value contained in variable XB.

SXAlh1h2  Inserts the value of XA in the byte of the buffer indicated by the 2-digit hex value h1h2.
Example

10  GOSUB 1000       Sub-routine line 1000.
20  SXDH01           Set XD for variable increment.
30  SXBH03           Set XB to current length.
40  SM010203         Set memory with length updated.
50  GOSUB 1000       Subroutine line 1000.

60  PRINT INPUT 0 TO STOP, 1 TO ADD BYTES
70  INPUT XE         Input choice for XE.
80  IXE=00 5000      If length = maximum, STOP. Note that 40 hex = 64 decimal)
85  IXB>40 5000      If length = maximum, STOP. Note that 40 hex = 64 decimal)

100 PRINT INPUT NEW BYTE TO ADD TO MEMORY.
110 INPUT XA         Increment variable.
120 SXB+XD           Adjust length.
130 SXB100           New position = length
140 SXAIXB           Unconditional jump to line 50.
150 GOTO 50          Display result of last update.
1000 DISPM
1010 DISPX
1020 RETURN          End of subroutine.
5000 PRINT BYE BYE!
### MESSAGE BUFFER BYTE EXTRACTION

**Description**
These commands enable you to extract one byte from the message buffer by specifying the location of the byte using a numeric variable or 2-digit hex value.

These commands can be used with the WAIT and STORE (WS) command to extract a byte from a frame received and stored by the Simulator.

**Type**
Statement

**Syntax**
- **SXAOXB**
  Extracts the byte at the location in the buffer indicated by the value contained in variable \( XB \), and stores the byte in \(XA\). \( XB \) cannot exceed 40 hex (64 decimal).
- **SXAOh1h2**
  Extracts the byte at the location in the buffer indicated by the 2-digit hex value \(h1h2\), and stores the byte in \(XA\). \(h1h2\) cannot exceed 40 hex.

**Example**
This example stores the value in the message **SXAO02** buffer. It also extracts the value at the second position in the message buffer and stores it in variable A. The variable A contains the value 06.

**SMH45063269F54C**
### ASSIGN MESSAGE BUFFER LENGTH

**Description**
These commands enable you to assign the value of byte 0, which indicates the length of the message buffer. By setting the buffer length, you can control how many bytes of the buffer are transmitted.

**Type**
Statement or Direct

**Syntax**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXA100</td>
<td>Sets the message buffer length to the value of variable XA. This is a special use of the S...I (define message) command, in that it defines only byte 0 (00) of the message buffer contents.</td>
</tr>
<tr>
<td>SXAO00</td>
<td>Extracts the current buffer length and stores the length in variable XA. This is a special use of the S...O (byte extraction) command, in that it extracts only byte 0 (00) of the message buffer contents.</td>
</tr>
</tbody>
</table>

**Example**
This example assigns the value 12 to variable XA, uses the variable to set the buffer length, and then displays the memory.

```
10 SXAH12
20 SXA100
30 DISPM
```

**Result:**

```
L = 018
M = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00
```
TEST MESSAGE BUFFER CONTENTS

Description  This command compares the contents of the message buffer to the byte and mask configuration in the command. If they are identical, control jumps to a specified line number. Otherwise, execution continues with the next statement.

Type  Statement

Syntax  ISXX/YY(XX/YY)(....) dddd

This command is used in a scenario where dddd is the line number to branch to if the test is true.

The first XX/YY pair corresponds to the first byte in the message buffer. The second pair corresponds to the second byte, and so on. XX is two Hexadecimal characters defining the expected binary configuration after masking.

YY is the 8 bit mask compared to its corresponding position in the frame. YY is ANDed with its corresponding byte position in the buffer. The result is compared to XX.

Example  This example checks the message buffer for a CALL packet.

100  IS00/00,00/00,00/00,00/00,0B/FF 130
110  PRINT NOT A CALL PACKET
120  GOTO 140
130  PRINT IT WAS A CALL PACKET
140  REM END OF SCENARIO
### TRANSMIT MESSAGE

**Description**
These commands transmit the frame, packet or data packet, as described below.

**Type**
Statement or Direct

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| **FM** | Transmits the frame, assigning the first byte of the message buffer (byte 1) to the first byte of the frame, and each successive byte of the buffer to the next bytes of the frame.  
Buffer byte 1 --- > TX frame byte 1  
Buffer byte N --- > TX frame byte N  
There is no automatic control. The buffer becomes the entire frame. |
| **PM** | Assigns the contents of the message buffer, excluding the message buffer length (byte 0), to the I-Field of a frame (byte 3 and following) and transmits it with an automatically controlled 8 bit address field, and modulo 8 N(S) and N(R).  
Buffer byte 1 --- > TX frame byte 3  
Buffer byte N --- > TX frame byte N + 3  
There are automatic frame controls, but no automatic packet controls. The buffer becomes the packet level and data, depending on buffer length. |
| **PUnDS** | Assigns the contents of the message buffer (beginning with byte 1) to the data portion of the I-Field, and transmits it from the pseudo-user n (n must be between 1 and 7) with automatic control of P(S) and P(R). If the pseudo-user n is not activated, ERROR 22 will be displayed.  
Buffer byte 1 --- > TX frame byte 6  
Buffer byte N --- > TX frame byte N + 6  
Automatic frame and packet level controls. Buffer becomes data only. |

**Example**
FM
TRACE BUFFER COMMANDS

Introduction

This section contains the commands that enable you to manipulate and use the trace buffer. Refer to Section 8.4 for a description of the SITREX trace buffer. Refer to Section 8.2 for a description of how the trace buffer interacts with the Automatic X.25 Simulator.

The trace buffer commands enable you to:

- Display the contents of the trace buffer
- Clear the trace buffer
- Save the contents of the trace buffer to disk
- Load a trace buffer file into memory
DISPLAY TRACE BUFFER

Description
The TPRINT command displays the transmitted and received frames by displaying the current contents of the trace memory. The command suffix allows you to change the interpretation of the trace as described below.

Notes
It is not recommended that you use these commands when a traffic generator is active, because the display slows the Simulator.

The data length of the trace capture is initialized to a value with the STRnn command.

Type
Statement

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPRINT</td>
<td>Displays all X.25 levels interpreted, with the data shown in hex.</td>
</tr>
<tr>
<td>TPRINTA</td>
<td>Displays all X.25 levels interpreted, with the data shown in ASCII.</td>
</tr>
<tr>
<td>TPRINTF</td>
<td>Displays all X.25 levels unintrepreted, with the frames shown entirely in hex.</td>
</tr>
<tr>
<td>TPRINTP</td>
<td>Only the frame level is interpreted. The I-Field of the I-Frames is shown in hex.</td>
</tr>
</tbody>
</table>

Example
This examples displays the contents of the trace buffer in hex.

TPRINTF

Result: T * 018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

The trace indicates T for transmission, * for good CRC, 018 for the length of 18 bytes, and then data (all zeros).
LOAD TRACE FILE FROM DISK

Description  The TLOAD command loads a trace file from disk to memory.

Type         Statement or Direct

Syntax       TLOAD

When you enter the TLOAD command, you are prompted for the drive and filename:

File Name:
Enter the name of the trace file in this format:

xfilename

where: x = 0 to load a trace file from the hard disk or x = 1 to load a file from the floppy drive.

Example      This example loads a trace file named TRACE A from the floppy drive.

TLOAD

File Name: 1TRACEA
PRINT TRACE BUFFER

Description
The LPRINT outputs the current contents of the trace memory to a printer or remote terminal. The command suffix allows you to change the interpretation of the trace as described below.

Notes
It is not recommended that you use these commands when a traffic generator is active, because the display slows the Simulator.

The data length of the trace capture is initialized to a value with the STRnn command.

Type
Statement

Syntax
LPRINT Displays all X.25 levels interpreted, with the data shown in hex.

LPRINTA Displays all X.25 levels interpreted, with the data shown in ASCII.

LPRINTF Displays all X.25 levels unintrepreted, with the frames shown entirely in hex.

LPRINTP Only the frame level is interpreted. The I-Field of the I-Frames is shown in hex.

Example
This example outputs the contents of the trace buffer in ASCII.
LPRINTA
### SAVE TRACE TO DISK

**Description**
This command saves the contents of the trace buffer to a file.

**Type**
Statement or Direct

**Syntax**

- `TSAVE"0filename"` Saves the trace file to the hard disk drive. Note that a blank space is not allowed between the 0 and the filename.

- `TSAVE"1filename"` Saves the trace file to the floppy disk drive. Note that a blank space is not allowed between the 1 and the filename.

**Example**
This example saves the contents of the current trace buffer to a file named TRAFFIC on the floppy disk drive:

`TSAVE"1TRAFFIC"`
## SET TRACE BUFFER ON/OFF

<table>
<thead>
<tr>
<th>Description</th>
<th>These commands enable you to make the trace buffer active or idle. For a complete description of the function of the trace buffer in SITREX, refer to Section 8.4.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td><strong>STON</strong> Sets the trace buffer ON. Stores transmitted and received frames in the trace buffer.</td>
</tr>
<tr>
<td></td>
<td><strong>STOF</strong> Sets trace buffer OFF. Stops the storage of transmitted and received frames in the trace buffer. This command does not clear the trace buffer.</td>
</tr>
<tr>
<td>Example 1</td>
<td>This example turns the trace buffer on so that it stores transmitted and received frames.</td>
</tr>
<tr>
<td></td>
<td><strong>STON</strong></td>
</tr>
<tr>
<td>Example 2</td>
<td>This example turns the trace buffer off. It becomes idle and does not store transmitted and received frames.</td>
</tr>
<tr>
<td></td>
<td><strong>STOF</strong></td>
</tr>
</tbody>
</table>
CLEAR TRACE BUFFER

Description
The TRACE command clears the trace buffer.

Type
Statement or Direct

Syntax
TRACE
Clears the trace buffer.

Example 1
In this example, TRACE is executed every 10 seconds, resulting in a circular trace.

```
10 STU0390    Sets user timer to 0390 hex.
20 IFTU 40    If timer expires, go to line 40.
30 GOTO 20    Returns to 20, checks timer again.
40 TRACE      Clears trace when time expires.
50 GOTO 10    Returns to 10, reset timer.
```

Example 2
This example uses TRACE with STON and STOF to capture transmitted and received I-frames in the trace buffer. This scenario captures only 10 frames after receiving a UA.

```
5 SWT FFFF    Sets watch-dog timer to FFFF.
6 SADRWT 130  Sets watch-dog jump address to line 130.
10 SC009      Set counter 0 to 9.
20 STOF       Set trace off.
30 TRACE      Clear trace buffer.
40 SELSE 50   Set WAIT jump address to 50.
45 SLON       Sets the link on.
50 WFUA       Wait for a UA.
60 SELSE 80   Set WAIT jump address to 80.
70 STON       Set trace on.
80 WTO0/01    Wait to receive an I-frame.
90 SCO-       Decrement counter 0.
100 IFCO 120  If counter 0 is elapsed, go to 120.
110 GOTO 80   Return to line 80.
120 STOF      Set trace off.
130 REM STOP
```
SET TRACE LENGTH

Description
This command defines the number of data bytes (0 - 255) displayed by the trace buffer. It is relevant only for the data portion of frames. It truncates the data field, enabling more frames to be displayed on the screen, and more to be stored in the trace buffer.

Type
Statement or Direct

Syntax
STRh1h2

where: h1h2 is a hexadecimal value in the range 0 to FF.

Example
This example displays only the first four bytes of data from each frame in the trace buffer.

STR04
DISPLAY AND PRINT COMMANDS

Introduction

This section contains the commands you use to display information on the screen or output it to a printer or remote device. The following types of information can be displayed and/or printed:

- Timer values
- Counter values
- Variable values
- Screen message and prompts
- Program files

See the Trace Buffer commands for information about displaying and printing the contents of the trace.
## DISPLAY USER PARAMETERS

<table>
<thead>
<tr>
<th>Description</th>
<th>The DISP command displays timers, user-counters, numeric variables, or the contents of the message buffer. The LDISP is the same as the DISP command, but output is to a printer or remote terminal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Statement or Direct</td>
</tr>
<tr>
<td>Syntax</td>
<td><strong>DISPT</strong> Displays the following timers in decimal: TU (User Timer), WT (current value of Watch-Dog Timer), and WDTOR (initial value of Watch-Dog Timer)</td>
</tr>
<tr>
<td></td>
<td>If you set TU to an initial value of FFFF (using <strong>STU h1h2h3h4</strong>), <strong>DISPT</strong> displays the time elapsed between the execution of the <strong>STU</strong> command and the <strong>DISPT</strong> command, in decimal.</td>
</tr>
<tr>
<td></td>
<td><strong>LDISPT</strong> Outputs timer values to a printer or a remote terminal.</td>
</tr>
<tr>
<td></td>
<td><strong>DISPC</strong> Displays the values of the seven user-defined counters in hex.</td>
</tr>
<tr>
<td></td>
<td><strong>LDISPC</strong> Outputs counter values to a printer or a remote terminal.</td>
</tr>
<tr>
<td></td>
<td><strong>DISPV</strong> Displays the values of the following variables: K (Frame Window size), N1 (Frame size), N2 (Number of retransmissions), T1 (Frame level timer), LCGN (Logical Channel Group Number)</td>
</tr>
<tr>
<td></td>
<td><strong>LDISPV</strong> Outputs above variable values to a printer or a remote terminal.</td>
</tr>
<tr>
<td></td>
<td><strong>DISPX</strong> Displays the current values of the 26 numeric variables (XA - XZ) in hex.</td>
</tr>
<tr>
<td></td>
<td><strong>LDISPX</strong> Outputs numeric variable values (XA - XZ) to a printer or a remote terminal.</td>
</tr>
</tbody>
</table>
DISPM

Display the length and contents of the message buffer. The length is shown in decimal; the contents in hexadecimal.

LDISPM

Outputs message buffer length and contents to a printer or a remote terminal.

Example

This example assigns values to variables and then displays the message buffer.

```
10 SMH010203
20 SXAH4C
30 SXAI02
40 DISPM
```

Result

```
L = 003
M = 01 4C 03
```
DISPLAY SCREEN MESSAGES AND PROMPTS

Description
The PRINT command displays a user-defined message on the Chameleon screen. During a SITREX scenario, these messages can be used to inform you of the status of a simulation.

This command cannot be used to display the value of variables. Refer to DISP.

Type
Statement

Syntax
PRINT text

This command must be terminated with a carriage return and not a colon (:), to avoid interpreting the colon as part of the printed string.

Example
This example waits for a SARM to be received and then displays the message SARM RECEIVED on the Chameleon screen.

10 SELSE 20
20 WFSARM
30 PRINT SARM RECEIVED
LIST PROGRAM FILE

Description
This command displays the program lines from the scenario currently in memory.

If the scenario is too long to fit on the screen, the display can be stopped by typing CTRL + S. To restart the listing, type CTRL + Q. The listing can be aborted by pressing ESCape.

Type
Direct

Syntax
LIST

Example
LIST
## PRINT PROGRAM FILE

<table>
<thead>
<tr>
<th>Description</th>
<th>This command outputs the program lines from the scenario currently in memory to a remote I/O device, terminal, or printer. The listing can be aborted by pressing ESCape.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Direct</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td>LLIST</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>LLIST</td>
</tr>
</tbody>
</table>
## OUTPUT MESSAGE TO PRINTER OR REMOTE DEVICE

**Description**
The LPRINT command outputs a user-defined message to a printer or remote terminal. During a SITREX scenario, these messages can be used to inform you of the status of a simulation.

This command cannot be used to print the value of variables. Refer to LDISP.

**Type**
Statement

**Syntax**
```
LPRINT
```

This command must be terminated with a carriage return and not a colon (:), to avoid interpreting the colon as part of the printed string.

**Example**
This example waits for a SARM to be received and then outputs the message SARM RECEIVED to a configured printer or remote terminal.

```
10 LPRINT This is being sent to the printer.
!RUN
```
8.8 SITREX ERROR CODES

Introduction

This section lists the error codes that may occur during SITREX simulation. The error messages are listed with their codes and interpretations.

ERROR NUMBER

The message "ERROR NUMBER" with a two-digit code indicates that a user input error has occurred. The two-digit codes can be interpreted as follows:

- **00**: First character incorrect.
- **04**: Illegal Line number (valid range is 0 to 9999).
- **15**: Attempt to re-assign a new LCN to a previously set pseudo-user.
- **16**: Attempt to give a previously assigned LCN to a new pseudo user.
- **18**: RETURN without GOSUB.
- **20**: Incomplete Loop (; before * nn).
- **21**: Line number specified does not exist.
- **22**: Attempt to send a data packet on a logical channel that has not been set up.
- **26**: Error in the call facility field of a call packet.
- **81-83**: No space for scenarios (memory full).

T nn

The message T with a two-digit code occurs when the frame level is on. Frame Error Coding is from the Automatic X.25 Simulator.

- **T00**: Reception of a frame with an I-field that exceeds N1.
- **T01**: Address unknown, frame ignored (Not Primary or Secondary address).
- **T02**: Response with poll final bit set when not solicited.
- **T03**: Response with poll final bit not set when solicited.
- **T04**: Response unknown, results in sending CMDR.
- **T05**: Incorrect length of response frame, results in CMDR.
- **T08**: Received unsolicited UA.
- **T09**: Incorrect Nr received.
- **T10**: Reject frame received.
- **T11**: RNR frame received.
- **T14**: Unknown command received.
- **T15**: Incorrect Ns received.
### SITREX-Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T16</td>
<td>I-Frame out of sequence and station not busy and no Tx.RNR requested and no prior Tx.REJ OR Tx.P/F set.</td>
</tr>
<tr>
<td>T17</td>
<td>I-frame out of sequence and station not busy and no Tx.RNR requested and no prior Tx.REJ and no Tx.P/F set.</td>
</tr>
<tr>
<td>T18</td>
<td>Received frame out of sequence with Tx.RR request.</td>
</tr>
<tr>
<td>T19</td>
<td>Received frame out of sequence.</td>
</tr>
<tr>
<td>T20</td>
<td>Internal error.</td>
</tr>
<tr>
<td>T21</td>
<td>Error which occurs when SITREX sends a disconnect and the device under test gives an unexpected response.</td>
</tr>
<tr>
<td>T22-24</td>
<td>Internal errors.</td>
</tr>
</tbody>
</table>

#### P nn

The message P followed by a two-digit code indicates that the packet level is on. The Packet Error Coding is from the Automatic X.25 Simulator.

- **P00**: Restart at the packet level.
- **P01**: Internal error.
- **P02**: Flow control anomaly (bad P(s) or P(r)).
- **P03**: Call or interrupt collision on a logical channel.

#### TABLE ERROR NUMBER

The message TABLE ERROR NUMBER followed by a two-digit code indicates an internal error:

- **01,02**
- **06,07**
- **0A,0B**
- **16**: This error is associated with high density traffic.
8.9 SITREX SAMPLE PROGRAMS

Introduction

This section contains examples of SITREX programs. In the samples, the Chameleon simulates a DTE and is connected to a modem line.

After establishing a connection at the frame level by typing SLON, virtual circuits can be established.

SITREX1

The first example is a series of commands that sends various types of packets. You will enter these commands, and then use the program management commands to save the program to disk, erase it from memory, load it from disk and list it on the screen, and run it.

!10 PU1CALL1234567801  Places a CALL from pseudo user 1 (PU1) to address 12345678, with a sub-address or port ID 01.
!20 PU1DAABCDEFGHJ  Sends Data "ABCDEFGHJ" in ASCII.
!30 PU1DH01 A0 A0 A0 FE  Send Data, "01 A0 A0 A0 FE" in hex.
!40 PU1DAABCDE•ABCD45  Send Data, "ABC DE" in ASCII and "ABCD45" in hex. The tilde (•) indicates that the Esc key was pressed to switch to hex data.
!50 PU1DA HELLO  Sends Data, "HELLO" in ASCII.
!60 PU1RSET0 0  Sends a RESET packet.
!70 PU1INT3 1  Sends an INTERRUPT packet.
!80 PU1CLEAR  Sends a CLEAR packet.  SITREX prompt.

!SAVE  Saves program entered above to disk.
FILENAME : ?1SITREX  System requests name for program file.  You respond with 1 (Drive B) and SITREX.
!NEW  Clears program from memory.  Since nothing is displayed, memory is cleared.
!LIST  Loads a program from disk to memory.  System requests name of file to load.  You respond with 1 (Drive B) and SITREX.
!LOAD  Displays program currently in memory.  Runs programs.  Error message 22 may result, indicating that the LCN was not set up prior to sending packets (see error codes in Section 8.8).  
FILENAME : ? 1 SITREX  
!LIST  
!RUN
SITREX2
This scenario sets up a CALL, transfers data, and sends a CLEAR. Each line is typed in when the ! prompt is displayed. The line numbers indicate that the command is to be stored in memory and executed in sequence. Lines can be entered in any order, as they will be re-ordered automatically according to the line number.

Entry Error Recovery
The command entered in line 70 contains a syntax error. When the RETURN key is pressed after the erroneous command, the line is re-displayed up to the occurrence of the error. The rest of the line can be re-entered correctly, or the fragment displayed may be deleted by using the back space key.

!10 REM CALL REQUEST, DATA TRANSFER AND CLEAR.

<table>
<thead>
<tr>
<th>Line</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>REM</td>
<td>You can enter comments here.</td>
</tr>
<tr>
<td>20</td>
<td>REM</td>
<td>Place a CALL from Psuedo-User 2.</td>
</tr>
<tr>
<td>30</td>
<td>PU2CALL19200039701</td>
<td>Set user timer to 01 F4 Hex.</td>
</tr>
<tr>
<td>40</td>
<td>STU01F4</td>
<td>If TU equals 0 go to line 70.</td>
</tr>
<tr>
<td>50</td>
<td>IFTU 70</td>
<td>Unconditional jump to line 50. Keep checking TU timer.</td>
</tr>
<tr>
<td>60</td>
<td>GOTO 50</td>
<td>Send data: syntax error.</td>
</tr>
<tr>
<td>70</td>
<td>PU2D1234567890</td>
<td>Syntax error corrected. Send data, &quot;1234567890&quot; in ASCII.</td>
</tr>
<tr>
<td>70</td>
<td>PU2DA1234567890</td>
<td>Set user timer to 01 F4 again.</td>
</tr>
<tr>
<td>80</td>
<td>STU01F4</td>
<td>If TU equals 0 go to line 110.</td>
</tr>
<tr>
<td>90</td>
<td>IFTU 110</td>
<td>Unconditional jump to line 90. Keep checking TU timer. Awaiting a response.</td>
</tr>
<tr>
<td>100</td>
<td>GOTO 90</td>
<td>Send a CLEAR packet.</td>
</tr>
</tbody>
</table>

Note
Timers are used to ensure that a response is received, as in this scenario. If the call in line 30 is cleared, and line 70 is then executed, an error will appear.
This scenario measures the time required to set up a call. Each time a WAIT command is executed, the Watch-dog Timer is set, allowing you to examine a specified logical channel during a user determined time.

For the Packet WAIT commands, the logical channel related to the Pseudo-User should be established before execution of the WAIT command. When a packet WAIT command is run, the use of Pseudo-User 7 is prohibited.

60 SWT1000 Set watchdog timer to 1000.
70 SADRWT 800 Jump to 800 if Watch-dog timer expires.
90 SELESE 110 Set ELSE to 110. Jump to 110 if frame expected by WAIT is incorrect.
100 PU5CALL19200039705 Set user timer to FFFF. Place a CALL from Psuedo-User 5.

110 WT00/00,00/00,00/00,00,00,00,00,00,00,OF/FF
120 DISPT Wait for a CALL CONFIRMATION.
150 STU01F4 Display timers.
160 IFTU 180 Set user timer to 01 F4 Hex.
170 GOTO 160 If TU equals 0 go to line 180.
190 SELSE 210 Unconditional jump to 160. Keep checking timer.
200 PU5CLEAR0 0 Set jump address of next WAIT to 210.
210 WPU5CCLEAR Send a CLEAR packet.
230 PRINT *** OK *** Wait for a CLEAR CONFIRMATION.
250 GOTO 60 Arrive here if correct response is received.

800 PRINT ***Watch-dog timer elapsed***
1000 REM  End of scenario

The following display shows a possible result of this scenario:

!RUN
TU = 000340MS WDTOR = 005000MS WDT = 004940MS
*** OK ***
TU = 000350MS WDTOR = 005000MS WDT = 004950MS
*** OK ***
TU = 000350MS WDTOR = 005000MS WDT = 004940MS
*** OK ***
TU = 000340MS WDTOR = 005000MS WDT = 004950MS
*** OK ***
*** JOB CANCELLED ***
!
!PU1CLEAR
The User Timer (TU) is initialized to FFFF while a CALL packet is transmitted. After the next command in the scenario, \texttt{WT00/00,00/00,00/00,00/00,01/FF} (with bit masks corresponding to a CALL CONFIRMATION packet), the timers are displayed (DISPT).

The TU is displayed when a CALL CONFIRMATION packet is received. The TU shows the time elapsed between a CALL packet transmission and the time a confirmation is received.

This scenario allows you to make several measurements. It is necessary to CLEAR the logical channel before returning the CALL. The scenario transmits the CLEAR, and waits for a CLEAR CONFIRMATION. It displays \texttt{** OK **} when it is received, indicating that the flow control is normal.
SITREX4

This scenario tests the DTE's ability to deal with invalid commands and responses.

2 PRINT START OF TEST. PLEASE SEND SABM.

GOTO 50 Go to line 50.
SELE 20 Sub-routine.
REM Set jump address for wait.
WFPSABM Wait for polled SABM.

30 PRINT POLLED SABM RECEIVED, WAITING FOR ANOTHER.

RETURN End of sub-routine.
TRACE Clear trace buffer.
SWTFFFF Set watchdog timer to 11 minutes.
SELE 90 Set jump address for watchdog.
SADRWT 1000 Set jump address for wait.
WFSABM Wait for SABM.

100 PRINT SABM RECEIVED. WAITING FOR POLLED SABM.

FH030000 Send I-frame.
GOSUB 10 Go to subroutine line 10.
FRR Send Receive Ready.
GOSUB 10 Send improper command.
FH0373 Go to subroutine line 10.
FREJ Send Reject.
GOSUB 10 Send RR frame exceeding.
FH01010 Maximum length.

210 PRINT TEST OK. LET DTE EXPIRE N2 VALUE.

GOTO 1500 End of test message.
PRINT WT ELAPSED Arrive here if watchdog expires.

1500 PRINT END OF TEST. LINK DOWN.

STU1000 Set user timer to 40 seconds.
IFTU 2020 If TU equals 0 go to line 2020.
GOTO 2010 Check TU again.
&1TEST2CF Load next test.
8.10 SIMULATOR REACTIONS

Introduction

Table 8.10-1 shows the actions performed by the DCE upon receiving packets in a given state of the DTE/DCE interface.

<table>
<thead>
<tr>
<th>PACKET FROM THE DTE</th>
<th>ANY STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACKET LENGTH SHORTER THAN 2 BYTES</td>
<td>DIAGNOSTIC #38 (26H)</td>
</tr>
<tr>
<td>INCORRECT FORMAT TYPE IDENTIFIER</td>
<td>DIAGNOSTIC #40 (28H)</td>
</tr>
<tr>
<td>OTHER</td>
<td>SEE TABLE 8.10-2</td>
</tr>
</tbody>
</table>

Table 8.10-1

- The DIAGNOSTIC packets are transmitted only by the DCE.
- The diagnostics are given in Hexadecimal coded values.
- The causes of a CLEAR transmitted by the DTE always have bit 7 of the CLEAR DIAGNOSTIC Field set to 1.
- None of the intermediate steps are shown for table clarity.
- The table indicates which state the Simulator will be in depending upon the previous events.

Packet Level

Table 8.10-2 shows the actions performed by the network in READY or RESTART state. The packet is received from the subscriber with an assigned logical channel number (LCN).
<table>
<thead>
<tr>
<th>STATE OF THE SIMULATOR: PACKET RECEIVED</th>
<th>READY (R1)</th>
<th>RESTART ALREADY SENT (R3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTART</td>
<td>RESTART CONFIRMATION</td>
<td>NO ACTION --&gt; R1</td>
</tr>
<tr>
<td>RESTART CONFIRMATION</td>
<td>NETWORK RESTART 01 --&gt; #11</td>
<td>READY STATE (R1)</td>
</tr>
<tr>
<td>INVALID PACKET</td>
<td>CLEAR 13 #21 --&gt; TABLE (3,4)</td>
<td>NO ACTION DISCARD PACKET --&gt; R3</td>
</tr>
<tr>
<td>PACKET OTHER THAN: DIAGNOSTIC</td>
<td>DIAGNOSTIC 24 --&gt; R1</td>
<td>DIAGNOSTIC 24 --&gt; R3</td>
</tr>
<tr>
<td>RESTART CONFIRMATION IF LCN = 0</td>
<td>TABLE 8.10-3</td>
<td>NO ACTION DISCARD PACKET --&gt; R3</td>
</tr>
<tr>
<td>OTHER</td>
<td>NO ACTION --&gt; READY STATE(R1)</td>
<td>NO ACTION --&gt; R3</td>
</tr>
</tbody>
</table>

Table 8.10-2

**DISCARD**

The Simulator will ignore the incoming packets and wait for PACKET RESTART time-out.

Table 8.10-3 shows the actions performed by the network in virtual CALL set-up or clear logical channel state.
### Table 8.10-3

<table>
<thead>
<tr>
<th>STATE OF THE SIMULATOR: PACKET RECEIVED:</th>
<th>READY</th>
<th>CALL ALREADY SENT</th>
<th>DATA TRANSFER</th>
<th>CLEAR ALREADY SENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>PROCESS RECEIVED CALL</td>
<td>PROCESS RECEIVED CALL</td>
<td>CLEAR 13 #17 --&gt; P7</td>
<td>DISCARD 2 --&gt; P7</td>
</tr>
<tr>
<td>CALL CONFIRMATION</td>
<td>CLEAR 13 #14 --&gt; P7</td>
<td>PROCESS RECEIVED CALL CONF. --&gt; P4</td>
<td>CLEAR 13 #17H --&gt; P7</td>
<td>DISCARD 2 --&gt; P7</td>
</tr>
<tr>
<td>CLEAR</td>
<td>CLEAR CONF. --&gt; P1</td>
<td>CLEAR CONFIRMATION --&gt; P1</td>
<td>CLEAR CONF. --&gt; P1</td>
<td>CLEAR CONFIRMATION --&gt; P1</td>
</tr>
<tr>
<td>CLEAR CONFIRMATION</td>
<td>CLEAR 13 #14 --&gt; P7</td>
<td>CLEAR 13 #16 --&gt; P7</td>
<td>CLEAR 13 #17 --&gt; P7</td>
<td>-- &gt; READY STATE (P1)</td>
</tr>
<tr>
<td>DATA RESET INTERRUPT FLOW CONTROL</td>
<td>CLEAR 13 #14 --&gt; P7</td>
<td>CLEAR 13 #16 --&gt; P7</td>
<td>Table 8.10-4</td>
<td>DISCARD 2 --&gt; P7</td>
</tr>
<tr>
<td>RESTART RESTART CONFIRMATION IF LCN # 0</td>
<td>CLEAR 13 #29 --&gt; P7</td>
<td>CLEAR 13 #29 --&gt; P7</td>
<td>Table 8.10-4</td>
<td>DISCARD 2 --&gt; P7</td>
</tr>
</tbody>
</table>

**DISCARD 1:** The simulator waits CALL time out.

**DISCARD 2:** The simulator waits CLEAR time out.

Table 8.10-4 shows the actions performed by the network upon data transfer and flow control on established logical channels.
<table>
<thead>
<tr>
<th>STATE OF THE SIMULATOR: PACKET RECEIVED:</th>
<th>DATA TRANSFER</th>
<th>RESET ALREADY SENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>RESET CONFIRMATION</td>
<td>RESET CONFIRMATION</td>
</tr>
<tr>
<td></td>
<td>-- &gt; D1</td>
<td>-- &gt; D1</td>
</tr>
<tr>
<td>RESET CONFIRMATION</td>
<td>RESET 05 #18</td>
<td>-- &gt; DATA TRANSFER</td>
</tr>
<tr>
<td></td>
<td>-- &gt; D3</td>
<td>(D1)</td>
</tr>
<tr>
<td>DATA INTERRUPT FLOW CONTROL</td>
<td>PROCESS FLOW CONTROL</td>
<td>DISCARD</td>
</tr>
<tr>
<td></td>
<td>-- &gt; D1</td>
<td>-- &gt; D3</td>
</tr>
<tr>
<td>RESTART</td>
<td>RESET 05 #29</td>
<td>DISCARD</td>
</tr>
<tr>
<td>RESTART CONFIRMATION IF LCN # 0</td>
<td>-- &gt; D3</td>
<td>-- &gt; D3</td>
</tr>
</tbody>
</table>

Table 8.10-4

**DISCARD:** The simulator waits for RESET timer to elapse.
Timer Charts

This section describes the Network and Subscriber Time-outs.

<table>
<thead>
<tr>
<th>TIMER VALUE</th>
<th>START TIMER</th>
<th>END TIMER</th>
<th>1ST TIME OUT</th>
<th>2ND TIME OUT</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>T10</td>
<td>60 s</td>
<td>TRANSMIT</td>
<td>RESTART CONF. --&gt; READY STATE</td>
<td>TRANSMIT RESTART 01</td>
<td>DIAG #34. --&gt; READY STATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESTART</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>180 s</td>
<td>TRANSMIT</td>
<td>RECEIVE CALL CONF. CALL CLEAR</td>
<td>TRANSMIT CLEAR 13 #31</td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>60 s</td>
<td>TRANSMIT</td>
<td>RECEIVE RESET CONF. RESET</td>
<td>TRANSMIT RESET 05 #33</td>
<td>TRANSMIT CLEAR 13 #33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>60 s</td>
<td>TRANSMIT</td>
<td>RECEIVE CLEAR CONF. CLEAR</td>
<td>TRANSMIT CLEAR 13 #32</td>
<td>TRANSMIT DIAG #32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## SUBSCRIBER TIME OUTS:

<table>
<thead>
<tr>
<th>TIMER VALUE</th>
<th>START TIMER</th>
<th>END TIMER</th>
<th>1ST TIME OUT</th>
<th>2ND TIME OUT</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>T20</td>
<td>180 s</td>
<td>TRANSMIT RESTART</td>
<td>RECEIVE RESTART CONF. --&gt; READY STATE</td>
<td>TRANSMIT RESTART 01 #34</td>
<td>NONE</td>
</tr>
<tr>
<td>T21</td>
<td>200 s</td>
<td>TRANSMIT CALL</td>
<td>RECEIVE CALL CONF. CALL CLEAR</td>
<td>TRANSMIT CLEAR 13 #31</td>
<td>NONE</td>
</tr>
<tr>
<td>T22</td>
<td>180 s</td>
<td>TRANSMIT RESET</td>
<td>RECEIVE RESET CONF. RESET</td>
<td>TRANSMIT RESET 05 #33</td>
<td>TRANSMIT CLEAR 13 #33</td>
</tr>
<tr>
<td>T23</td>
<td>60 s</td>
<td>TRANSMIT CLEAR</td>
<td>RECEIVE CLEAR CONF. CLEAR</td>
<td>TRANSMIT CLEAR 13 #32</td>
<td>DIAG #32</td>
</tr>
</tbody>
</table>
8.11 FLOW CHARTS

Overview

The flow charts on the following pages provide a general overview of SITREX. The code numbers for the packets shown represent both the cause and diagnostic.

For example: RESET 05 #01

where 05 is the cause code and 01 is the diagnostic code.

For more information, see the CCITT X.25 Specifications.
CALL RECEIVED PROCESS
CALLED ADDRESS

CALL RECEIVED

Address Recognized?

YES

Pseudo User Free?

YES

CLEAR 00 #40

NO

CLEAR 01 #40

CLEAR 09 #40

Terminal Present?

YES

Data Packet Length?

DATA LENGTH = 0?

YES

CLEAR 13 #40

NO

Facilities Correct?

YES

NO

CLEAR 03 #41

CLEAR 03 #42

CLEAR 03 #49

Supported Code?

YES

Valid Code Parameters?

YES

No Duplicated Code?

YES

Invalid Code Combinations?

NO

NO

Figure 8.11-1: Flow Chart 1.
CALL RECEIVED PROCESS
CLOSED USER GROUP SELECTION

Figure 8.11-2. Flow Chart 2
CALL RECEIVED PROCESS
FAST SELECT

Figure 8.11-3: Flow Chart 3
RECEIVED CALL PROCESS
WINDOW SIZE PARAMETERS

4

Window Size Requested?

NO YES

Window Size Subscriber?

NO YES

CLEAR 83 #41

Default Window Size Values

Requested Window Size Values

5

Figure 8.11-4: Flow Chart 4
RECEIVED CALL PROCESS
PACKET LENGTH PARAMETERS

Figure 8.11-5: Flow Chart 5
CALL RECEIVED PROCESS
TRANSMISSION OF CALL CONFIRMATION

Figure 8.11-6: Flow Chart 6
RECEPTION OF CALL CONFIRMATION

The logical channel goes to either a Data Transfer State with flow control parameters, or the negotiated parameters (Flow Charts 4, 5, and 6) are transmitted in the Call packet, if the Call Confirmation packet does not explicitly contain the determined values (and if the transmitted values are correct).

Figure 8.11-7: Flow Chart 7
DATA RECEIVED PROCESS

DATA PACKET Received

P(S) in sequence?

YES

NO

RESET 05 #01

P(R) in sequence?

YES

NO

RESET 05 #02

Receive data ready to receive again

Figure 8.11-8: Flow Chart 8
RECEIVE FLOW CONTROL PROCESS
PACKET RECEIVE READY

- PRR Received
  - P(R) in sequence? NO → RESET 05 #02
  - YES → Permission to Transmit

PACKET RECEIVE NOT READY

- PRNR Received
  - P(R) in sequence? NO → RESET 05 #02
  - YES → Stop Transmission

Figure 8.11-9: Flow Chart 9
RECEIVE FLOW CONTROL PROCESS
REJECT PACKET

- Reject Packet Received
  - P(R) in sequence?
    - NO: RESET 05 #02
    - YES: P(R) in sequence?
      - NO: RESET 05 #02
      - YES: P(R) in sequence?
        - NO: RESET 05 #02
        - YES: Re-transmit Data packet

Figure 8.11-10: Flow Chart 10
INDEX

@ (ARRAY), 3.2-11, 3.5-2
ABORTTRAN, 4.3-2
ABS, 3.5-35.8
ACQUISITION BUFFER, 3.2-2
ARITHMETIC OPERATORS, 3.2-6
ARRAY, 3.2-11, 3.5-2
ASCS, 3.5-4
ASCII
  Code conversion, 3.1-2
  String Commands, 3.4-7
ASYNC (Chameleon)
  Applications, 1.2-3, 7.1-1
  Commands:
    BREAK, 7.5-2
    CRC16, 7.5-3
    FOXMESS, 7.5-4
    FRAMING, 7.5-5
    LENGTH, 7.5-6
    LRC, 7.5-7
    PARITY, 7.5-8
    REC, 7.5-9
    RXBREAK, 7.5-10
    TPRINT, 7.5-11
    TRAN, 7.5-12
  Description, 1.2-2
  Mnemonic Table, 7.3-1
  Parameter Set-Up Menu, 7.2-1
  Read-Only Variables, 7.4-1
  Reception Commands, 7.4-1
  Sample Programs, 7.6-1
  Subsets, 1.2-1
  Transmission Commands, 7.4-1
ATIMES, 3.5-5
AUTO, 3.5-6
BADTRAN, 4.3-3
BASIC (Chameleon)
  Abbreviations, 3.3-4
  Command types, 3.3-1
  Commands:
    @ (ARRAY), 3.2-11, 3.5-2
    ABS, 3.5-3
    ASCS, 3.5-4
    ATIMES, 3.5-5
    AUTO, 3.5-6
    BCDS, 3.5-7
    BLK, 3.5-8
    BLKHLL, 3.5-9
    BLKREV, 3.5-10
    BLKUND, 3.5-11
    CALL, 3.5-12
    CHAIN, 3.5-15
    CHRS, 3.5-16
    CLEAR, 3.5-17
    CLOSE, 3.5-18
    CLS, 3.5-19
    COUPLER, 3.5-20
    DECS$, 3.5-21
    DEFINE, 3.2-4, 3.5-22
    DEL, 3.5-23
    DELETE, 3.5-24
    DISP, 3.2-3, 3.5-25
    EBCS$, 3.5-26
    EDIT, 3.5-27
    EOF, 3.5-29
    EAEOE, 3.5-30
    EAEOE$, 3.5-31
    ERASE, 3.5-32
    EXIT, 3.5-33
    FDEFINE, 3.5-34
    FILES, 3.5-35
    FLIST, 3.5-36
    FLOAD, 3.5-37
    FLUSH, 3.5-38
    FOR, 3.5-39
    FREE, 3.5-40
    FSASVE, 3.5-41
    GOSUB, 3.5-42
    GOTO, 3.5-43
    HEXS, 3.5-44
    HEX>, 3.5-45
    HLF, 3.5-46
    HLFUND, 3.5-47
    IF, 3.5-48
    INKEYS, 3.5-50
    INPUT, 3.5-51
    $INPUT, 3.5-53
    INS, 3.5-54
    INSTR, 3.5-55
    KILL, 3.5-56
    LEFTS$, 3.5-57
    LEN, 3.5-58
    LET, 3.5-59
    LFILES, 3.5-60
    LFLIST, 3.5-61
    LIST, 3.5-62
    LLIST, 3.5-63
    LMLIST, 3.5-64
    LOAD, 3.5-65
    LPRINT, 3.5-66
    MENU, 3.5-67
    MERGE, 3.5-68
    MIDS, 3.5-69
    MLIST, 3.5-70
    MLOAD, 3.5-71
    MSAVE, 3.5-72
    NEW, 3.5-73
DEFINE, 3.2-4, 3.5-22, 4.3-5, 4.5-7, 5.3-5
DEFSUB, 4.3-6
DELETE, 3.5-23
DISCONNECT, 4.6-10
DISPF, 3.2-3, 3.5-22
DMATCH, 4.6-11
DMI (See FRAMEM DMI, Section 4.6)
DTIMERS, 4.6-12

EBCS, 3.5-26
EBCDIC
  Code Conversion, 3.1-2
  String Commands, 3.4-7
EDIT, 3.5-27
EOF, 3.2-10, 3.5-29
ERAEOL, 3.5-30
ERAEOS, 3.5-31
ERASE, 3.5-32
ERROR HANDLING
  BASIC, 3.3-5
  SITREX, 8.8-1
EXIT, 3.5-33
EXTEND, 4.3-7, 5.6-13

FDEFINE, 3.5-34
FILES, 3.5-35
FILL, 4.5-8
FList, 3.5-36
FLOAD, 3.5-37
FLUSH, 3.5-38
FOR, 3.5-39
FOXMESS, 7.5-4
FRAMEM
  Addressing Commands, 4.2-2
  Applications, 1.2-3, 4.1-1
  Commands:
    ABORTTRAN, 4.3-2
    BADTRAN, 4.3-3
    CRC, 4.3-4
    DEFINE, 4.3-5
    DEFSUB, 4.3-6
    EXTEND, 4.3-7
    GET, 4.3-8
    MOD, 4.3-9
    NORM, 4.3-10
    PUT, 4.3-11
    REC, 4.3-12
    RXADDR, 4.3-13
    RXC R, 4.3-14
    RXDIAG, 4.3-15
    RXFCtl, 4.3-16
    RXFLEN, 4.3-17
    RXN(R), 4.3-18
    RXN(S), 4.3-19
    RXP F, 4.3-20
    RXRFCTL, 4.3-21
    RXRP F, 4.3-22
    RXV(R), 4.3-23
    RXV(S), 4.3-24
    STATUS, 4.3-25
    TPRINT, 4.3-26
    TRAN, 4.3-27
    TXADDR, 4.3-29
    TXC R, 4.3-30
    TXDIAG, 4.3-31
    TXFCTL, 4.3-32
    TXI FIELD, 4.3-33
    TXN(R), 4.3-35
    TXN(S), 4.3-36
    T XP F, 4.3-37
    TXRFCTL, 4.3-38
    TXRP F, 4.3-39
    TXV(R), 4.3-40
    TXV(S), 4.3-41
  Description, 1.2-2, 4.1-1
  I-frame, 4.1-1, 4.1-2
  Mnemonic Commands, 4.2-2
  Reception Variables, 4.2-3
  Subsets, 1.2-1
  Transmission Variables, 4.2-3
FRAMEM DMI
  Call Setup Commands, 4.6-3
  Commands:
    CAUSE, 4.6-5
    CHADMIN, 4.6-6
    CONNECT, 4.6-7
    DCALLED, 4.6-8
    DCALLING, 4.6-9
    DISCONNECT, 4.6-10
    DMATCH, 4.6-11
    DTIMERS, 4.6-12
    GLARE, 4.6-15
    MATCH, 4.6-16
    OUTNUM, 4.6-17
    RESET, 4.6-18
    RESPTIME, 4.6-19
    STATE, 4.6-20
    STATUS, 4.6-21
    TPRINT, 4.6-22
  Read-Only Variables, 4.6-3
  Sample Programs, 4.6-21
  Setting up, 4.6-1
FRAMEM HDLC
  Commands, 4.4-6
INDEX

TPRINT, 4.4-7
Configuration, 4.4-3
Mnemonic Table, 4.4-5
Parameter Set-Up Menu, 4.4-4
Transparent Bisync Menu, 4.4-3
Sample Programs, 4.4-8

FRAME MEM LAPD
Capabilities, 4.5-2
Commands:
  DEFINE, 4.5-7
  FILL, 4.5-8
  RXCR, 4.5-9
  RXSAPI, 4.5-10
  RXTEI, 4.5-11
  TPRINT, 4.5-12
  TXCR, 4.5-13
  TXSAPI, 4.5-14
  TXTEI, 4.5-15
Configuration, 4.5-4
LAPD Frame, 4.5-1
Mnemonic Table, 4.5-2
Parameter Commands, 4.5-5
Parameter Set-Up Menu, 4.5-3
Reception Variables, 4.5-6
Sample Programs, 4.5-16
Transmission Variables, 4.5-6

FRAME MEM SDLC
Commands, 4.4-6
  TPRINT, 4.4-7
Configuration, 4.4-3
Mnemonic Table, 4.4-5
Parameter Set-Up Menu, 4.4-4
Transparent Bisync Menu, 4.4-3
Sample Programs, 4.4-8

FRAMING, 7.5-5
FREE, 3.2-10, 3.5-40
FRELNK, 5.7-5, 5.8-5
FRSTAT, 5.7-6, 5.8-6
FSAVE, 3.5-41
FUNCTION KEYS
  Commands, 3.4-4
  Status Key, 2.1-1
  Main Menu, 2.1-1 to 2.1-4

GET, 4.3-8
GLARE, 4.6-15
GOSUB, 3.5-42
GOTO, 3.5-43
HDL C
  FRAME MEM (Section 4.4)
  SIMP-L (Section 5.4)
  Simulation, 1.2-1
HEX$, 3.5-44

HEX>, 3.5-45
HEXADECIMAL
  Code Conversion, 3.1-2
  String Functions, 3.4-7
HLF, 3.5-46
HLFUND, 3.5-47

IDLE, 6.5-3
IF, 3.5-48
INKEYS, 3.5-50
INPUT, 3.5-51
SINPUT, 3.5-53
INS, 3.5-54
INSTR, 3.5-55

KILL, 3.5-56

LAPD
  FRAME MEM (Section 4.5)
  SIMP-L (Section 5.6)
  Simulation, 1.2-1
LEFTS, 3.5-57
LEN, 3.5-58
LENGTH, 5.3-6, 7.5-6
LET, 3.5-59
LFILES, 3.5-60
LFLIST, 3.5-61
LIST, 3.5-62
LLIST, 3.5-63
LMLIST, 3.5-64
LNKSTAT, 5.3-7, 5.4-8, 5.5-9, 5.6-14, 5.7-2
LOAD, 3.5-65
LOGICAL OPERATORS
  Bitwise, 3.1-2, 3.2-6
  Boolean, 3.1-2, 3.2-6
LRC, 6.5-4, 7.5-7
LRDISPF, 5.3-8
LTDISPF, 5.3-9
LTPRINT, 3.5-66

MATCH, 4.6-16
MEMORY
  Programming size, 3.1-2
  Trace Buffer, 3.2-2
MENU, 3.5-67
MENUS
  Main, 2.1-1 to 2.1-3
  Parameter Set-Up (Section 2.2)
  Save Parameters (Section 2.3)

Index-4 • Version 4.7
INDEX

MERGE, 3.5-68
MIDS, 3.5-69
MLIST, 3.5-70
MLOAD, 3.5-71

MNEMONICS
ASYNC, 3.2-4, 7.3-1
BASIC Commands, 3.4-5
BSC, 3.2-4, 6.3-1
FRAME M, 3.2-4, 4.2-2
FRAME M HDLC, 4.4-5
FRAME M LAPD, 4.5-5
FRAME M SDL C, 4.4-5
SIM P L HDLC, 5.4-6
SIM P L LAPD, 5.6-32
SIM P L SDL C, 5.5-7
Usage, 3.1-3, 3.2-4
MOD, 4.3-9, 5.6-16
MSAVE, 3.5-72

MULTI-LINK SIM P L (Section 5.7)
Commands, 5.7-3 thru 5.7-4
Frame status bytes, 5.7-3
General Notes, 5.7-2
Introduction, 5.7-1
Link Selection, 5.7-1
Sample Program, 5.7-15
Variables, 5.7-2

NEW, 3.5-73
NEXT, 3.5-74
NORM, 4.3-10
NR M, 3.5-75
NSI, 5.5-10

NUMERIC VARIABLES
Description, 3.2-8
Usage, 3.2-8

OPEN, 3.5-76

OPERATORS
Arithmetic, 3.2-6
Logical, 3.1-2, 3.2-6
Precedence, 3.2-7
OUTN U M, 4.6-17

PARAMETER SET-UP MENUS
Async, 7.2-1
BSC, 8.2-1
Description (Section 2.2)
DEFAULT file, 2.2-1, 2.2-3
FRAME M HDLC, 4.4-2
FRAME M LAPD, 4.5-3
FRAME M SDL C, 4.4-2

PRINT, 3.5-78
PUT, 4.3-11

RDISP F, 3.2-3, 5.3-10
RE A D, 3.5-80
READ-ONLY VARIABLES
EOF, 3.2-10, 3.5-29
FREE, 3.2-10, 3.5-40
SIZE, 3.2-10, 3.5-95
TFREE, 3.2-10, 3.5-98
TIME, 3.2-10, 3.5-99
Usage, 3.2-10, 3.4-5
REC, 3.5-81, 4.3-12, 5.3-11, 6.5-5, 7.5-9
RECLNK, 5.7-7, 5.8-7
REM, 3.5-82
RE S ET, 4.6-18
RE SEQ, 3.5-83
RESPTIME, 4.6-19
RETURN, 3.5-85
REV, 3.5-86
REVHLF, 3.5-87
REVUND, 3.5-88
RIGHTS, 3.5-89
RND, 3.5-90
RTRANS, 5.8-8
RUN, 3.5-91
RXADDR, 4.3-13
RXBREAK, 7.5-10
RXCR, 4.5-9
RXC/R, 4.3-14
RXDIAG, 4.3-15
RXFCTL, 4.3-16
RXF RLEN, 4.3-17
RXL ENGTH, 6.5-6
RXN(R), 4.3-18
RXN(S), 4.3-19
RXP/F, 4.3-20
RXRFCTL, 4.3-21
RXR/P/F, 4.3-22
RX SAPI, 4.5-10
RXTEI, 4.5-11
RXV(R), 4.3-23
RXV(S), 4.3-24
SAVE, 3.5-92
INDEX

TRXIDR, 5.6-30
UNEXTEND, 5.6-31

Configuration:
Frame Level, 5.6-28
Checklist, 5.6-33
Set-Up Menu, 5.6-7

Converting Programs
to Extended LAPD, 5.6-42
Extended SIMP L LAPD, 5.6-1

Frame Specifications, 5.6-4 to 5.6-6
Mnemonic Table, 5.6-33

Multi-Link SIMP/L
Commands, 5.7-3 thru 5.7-4
Frame status bytes, 5.7-3
General Notes, 5.7-2
Introduction, 5.7-1
Link Selection, 5.7-1
Sample Program, 5.7-15
Variables, 5.7-2

Packet Status Byte, 5.6-3
Parameter Commands, 5.6-10
Parameter Set-Up Menu, 5.6-8
Q.931 Message format, 5.6-32
Read-Only Variables, 5.6-10
Sample Program, 5.6-34
Status Control Byte, 5.6-2
Transmission Commands, 5.6-10
XID Frames, 5.6-4

SIMP/L SDLC
Applications, 1.2-3, 5.5-1
Commands:
LNKSTAT, 5.5-9
NSI, 5.5-10
SET T1, 5.5-11
SET N2, 5.5-11
SET T1, 5.5-11
SET ADDR, 5.5-11
STATUS, 5.5-12
TEST, 5.5-13
TRPINT, 5.5-14
XID, 5.5-15
XIDFLD, 5.5-16

Configuration, 5.5-5
Frame Specifications, 5.5-2
Mnemonics, 5.5-7
Read-Only Variables, 5.5-8
Reception Commands, 5.5-8
Parameter Set-Up Menu, 5.5-4
Sample Programs, 5.5-17
Transmission Commands, 5.5-8

SIMP/L V.120 (Section 5.8)
Introduction, 5.8-1
Commands, 5.8-3

FRFLNK, 5.8-5
FRSTAT, 5.8-6
RECLNK, 5.8-7
RTRAN, 5.8-8
SET LINK, 5.8-9
SET LLI, 5.8-10
STATE, 5.8-11
STATUS, 5.8-12

Sample Program, 5.8-13

SIMULATION
Description, 1.1-1
Applications, 1.1-1
X.25, 1.1-1, 1.2-1
SNA, 1.1-1
BSC, 1.1-1
ISDN, 1.1-1

SIMULATORS
Description, 1.2-1 to 1.2-3
Roadmap, 2.1-5
Selecting, 2.1-1 to 2.1-4

SITREX
Applications, 1.2-3, 8.1-1
Automatic X.25 Simulator
Definition, 8.1-2, 8.1-3
Access, 8.2-1
Commands, 8.2-2
Pseudo-Users, 8.2-3
Reactions, 8.10-1 to 8.10-6
NetworkTimeouts, 8.10-5
Subscriber Timeouts, 8.10-6

Circuit Management, 8.1-11
Command Mode (Section 8.5)
Commands (Alphabetical List), 8.6-1 to 8.6-6
Conditional Jump, 8.7-45
Controlling X.25 Levels, 8.2-4
Counters, 8.1-10
Description, 1.2-1, 1.2-2, 6.1-1

Display and Print Commands:
Display Trace buffer, 8.7-67.
Print Trace buffer, 8.7-69.
Display Parameters, 8.7-75
Display Messages, 8.7-77
Display program, 8.7-78
Print program, 8.7-79
Print Messages, 8.7-80

Environment, 8.1-2
Error Codes, 8.8-1
Errors, syntax, 8.5-6

Flow Charts
Called Address, 8.11-2
Closed User Group, 8.11-3
Fast Select, 8.11-4
Window Size Parameters, 8.11-5

Index 7 • Version 4.7
INDEX

Packet Length Parameters, 8.11-6
Call Confirmation, 8.11-7
Call Confirmation Reception, 8.11-8
Data Received, 8.11-9
Packet Receive Ready, 8.11-10
Reject Packet, 8.11-11

Frame Level Commands:
  Information Frames, 8.7-7
  Numbered Commands, 8.7-5
  Numbered Responses, 8.7-6
  Unnumbered Commands, 8.7-3
  Unnumbered Responses, 8.7-4
  User-Defined Frames, 8.7-2

Loop Command, 8.7-44
Memory, 8.1-10

Menus
  Facilities, 8.3-6 to 8.3-7
  Frame Level, 8.3-4 to 8.3-5
  Packet Level, 8.3-4 to 8.3-5
  Physical Level, 8.3-2 to 8.3-3

Message Buffer Commands:
  Define Messages, 8.7-60
  Extract Byte, 8.7-62
  Assign Length, 8.7-63
  Test Contents, 8.7-64
  Wait and Store, 8.7-36
  Transmit, 8.7-65

Numeric Variable Commands:
  Keyboard Input, 8.7-58
  Operations, 8.7-56
  Rotate, 8.7-57
  Shift, 8.7-57
  State, 8.1-10
  Test, 8.7-59

Packet Level Commands:
  Call, 8.7-10
  Call Confirmation, 8.7-10
  Clear, 8.7-13
  Clear Confirmation, 8.7-13
  Data, 8.7-14
  Diagnostic, 8.7-15
  Interrupt, 8.7-13
  Interrupt Confirmation, 8.7-13
  Reset, 8.7-13
  Reset Confirmation, 8.7-13
  Restart, 8.7-12
  Restart Confirmation, 8.7-12
  Supervisory, 8.7-11
  User-Defined Packets, 8.7-9

Parameter Commands:
  Counters, 8.7-31
  Data Packet Length, 8.7-25
  Force Link On, 8.7-21
  Frame Level On Off, 8.7-17

Frame Window, 8.7-24
Link On-Off, 8.7-20
LCGN, 8.7-27
Packet Level On-Off, 8.7-19
Packet Window, 8.7-24
Permanent Virtual Circuit (PVC), 8.7-33
Primary Address, 8.7-23
Pseudo-User Type, 8.7-32
Secondary Address, 8.7-23
Set Interface Leads, 8.7-28
State Variables, 8.7-26
Switched Virtual Circuit (SVC), 8.7-33
Test Interface Leads, 8.7-29
Timers, 8.7-30
Transmit CRC, 8.7-22

Program Management Commands:
  CHAIN (&), 8.7-49
  LOAD, 8.7-50
  REMARK, 8.7-51
  NEW, 8.7-52
  RUN, 8.7-53
  SAVE, 8.7-54

Programming
  Command Groups, 8.5-2
  Command Mode, 8.5-1
  General Syntax, 8.5-3 to 8.5-5
  Syntax Errors, 8.5-6

Reinitialization Commands:
  EXIT, 8.7-47
  HALT, 8.7-47
  ESCape key, 8.7-47

Sample Programs, 8.9-1
Timers, 8.1-10

Trace Buffer
  Activating, 8.4-1
  Clear Trace, 8.7-72
  Commands, 8.4-2
  Description, 8.1-4, 8.4-1
  Display, 8.7-67
  Interpretation, 8.4-4 to 8.4-8
  Load File, 8.7-68
  Print contents, 8.7-69
  Save Trace, 8.7-70
  Set Length, 8.7-73
  Set Trace On Off, 8.7-71

Unconditional Jump, 8.7-46

Wait Commands:
  Introduction, 8.7-34
  Set jump addresses, 8.7-41
  Store byte mask, 8.7-38
  Store frame, 8.7-36
  Store packet, 8.7-36

Index-8 • Version 4.7
Wait for byte mask, 8.7-38
Wait for frame, 8.7-36
Wait for packet, 8.7-36
Watchdog Timer, 8.7-42
X.25 implementations
 CCITT, 8.1-1
 DATAPAC, 8.1-1
 NTT, 8.1-1
 TRANS PAC, 8.1-1
SIZE, 3.2-10, 3.5-95
SLOF, 5.3-13
SLON, 5.3-14
STATE, 4.6-20, 5.7-12, 5.8-11
STATUS, 4.3-25, 4.6-21, 4.6-21, 5.3-15,
 5.4-12, 5.5-12, 5.6-25, 5.7-13, 5.8-12
STOP, 3.5-96
STRINGS
 Code Conversion, 3.1-2
 Commands, 3.4-6
 Variables, 3.2-9
TDISP, 3.2-3, 5.3-16
TEST, 3.5-97, 5.5-13
TFREE, 3.2-10, 3.5-98
TIME, 3.2-10, 3.5-99
TIMES, 3.5-100
TIMERS, 3.1-2, 3.2-5
TLOAD, 3.5-101
TPRINT, 3.5-102, 4.3-26, 4.4-7, 4.5-12,
 4.6-22, 5.3-17, 5.4-13, 5.6-26, 5.7-14, 6.5-7, 7.5-11
TRACE BUFFER
 Commands, 3.4-7
 Defined, 3.1-3
 Display, 3.2-3
 SITREX Trace, 8.4-1 to 8.4-8
TRAN, 6.5-8, 4.3-27, 5.3-18, 7.5-12
TROFF, 3.5-103
TRON, 3.5-104
TRUI, 5.6-28
TRXIDC, 5.6-29
TRXIDR, 5.6-30
TSAVE, 3.5-105
TUTORIAL, BASIC, 3.3-6
TXADDR, 4.3-29
TXBUFFER, 6.5-9
TXCR, 4.5-13
TXC/R, 4.3-30
TXDIAG, 4.3-31
TXFCTL, 4.3-32
TXFIELD, 4.3-33
TXN(R), 4.3-35
TXN(S), 4.3-36
TXP/F, 4.3-37
TXRFTL, 4.3-38
TXRP/F, 4.3-39
TXSAPI, 4.5-14
TXSTATUS, 6.5-10 to 6.5-13
TXTEI, 4.5-15
TXV(R), 4.3-40
TXV(S), 4.3-41
UND, 3.5-106
UNEXTEND, 5.6-31
V.120 SIMPCL (Section 5.8)
 Introduction, 5.8-1
 Commands, 5.8-3
 FRELNK, 5.8-5
 FRSTAT, 5.8-6
 RECLNK, 5.8-7
 RTRAN, 5.8-8
 SET LINK, 5.8-9
 SET LLI, 5.8-10
 STATE, 5.8-11
 STATUS, 5.8-12
 Sample Program, 5.8-13
VAL, 3.5-107
VARIABLES
 Numeric, 3.2-8
 Read-Only, 3.2-10, 3.4-5
 String, 3.2-8
WRITE, 3.5-108
XID, 5.5-15
XIDFLD, 5.5-16
XYPLOT, 3.5-109