



Tattletale Model 8

Installation and Operation Manual

For use with TxBASIC

Onset Computer Corporation
470 MacArthur Blvd., Bourne, MA 02532
PO Box 3450, Pocasset, MA 02559-3450

Tel: (508) 759-9500
Fax: (508) 759-9100
www.onsetcomp.com

P/N MAN-TT8TXB



Copyright

© 1999, Onset Computer Corporation

All rights reserved. No part of this manual may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, optical, magnetic, or otherwise, without authorization from Onset Computer Corporation.

Trademarks

Tattletale and CrossCut are trademarks of Onset Computer Corporation. Aztec C is a trademark of Manx Software Systems. MS-DOS is a trademark of Microsoft. UNIX is a trademark of AT&T Bell Laboratories. PC-DOS is a trademark of IBM.

Warranty

Within one year after delivery, Onset will repair or at its option replace, without charge, any of its products found to have a manufacturing defect. Boards damaged by customer error or negligence, or that have failed after the one year period, may be returned for evaluation.

Replacement Policy

New replacements for damaged Tattletale products will be made available as long as the product has not been discontinued. The cost is approximately 1/2 the quantity one price. The replacement is warranted for one year. A customer may request a replacement for any reason as long as the damaged board can be returned. Onset may suggest replacing an item submitted for repair if the cost of the repair will exceed the cost of the replacement.

Repair Policy

Onset will attempt to repair Tattletale products returned with an Onset RMA number. Estimates will not be given but in no case will the customer be charged more than the cost of a new replacement. After the item is repaired or replacement is recommended the customer will be notified of the price. Repairs or replacements will not be shipped until a valid purchase order is received. Electronic items which have been repaired may be more prone to future failure than new items. Onset does not guarantee the appropriateness or necessity of any repair. Repairs will usually be finished in less than 2 weeks.

ASAP Repair Policy

Repairs will be started the same day they are received if the following conditions are met:

- *The damaged item is clearly labeled "ASAP repair requested".
- *The damaged item is accompanied with an open purchase order and an Onset RMA number.
- *The customer's account is not over due.
- *The damaged item is received before noon.
- *The appropriate Onset Computer employees are present on the day of receipt.

Onset will do its best to repair the item the same day it is received, however, due to circumstances beyond our control this may not always be possible. ASAP repairs carry a higher retest and troubleshooting charge. ASAP repairs will be returned UPS red or Fed X priority 1 at the customer's expense unless another option is requested.

Disclaimer

Onset makes no warranties, either express or implied, regarding the Tattletale, its merchantability, or its fitness for any particular purpose. The exclusion of implied warranties is not permitted by some states. As such, the exclusion may not apply to you. The Tattletale and its development boards are not authorized for use as critical components in life support, or other medical devices or systems without the express written approval of the President of Onset Computer Corp.

Contents

<u>Title</u>	<u>Page</u>
Section 1 - Introduction to the Model 8	
Quick Start Information	1-1
Welcome to the Tattletale Model 8	1-2
Onset Computer Corporation.....	1-3
Conventions used in this Document	1-4
Where to Start	1-5
What You Should Know	1-5
Warnings and Precautions	1-5
Additional Information Resources.....	1-5
Documentation	1-5
Technical Support	1-6
Development Software Description	1-7
IBM Software	1-7
Macintosh Software	1-7
Getting Started	1-7
Tattletale Model 8 Development Kit Contents.....	1-7
What to do if something is Missing or Broken upon Arrival	1-9
Tools Required to Connect and Test the Tattletale.....	1-9
Safety Precautions	1-9
Section 2 - How to Connect and Setup the Model 8	
The Prototyping Board	2-1
Attaching the Tattletale Model 8 to the IO-8 Prototyping Board	2-1
Installing a Temporary Sensor onto the Prototyping Board for Testing.....	2-1
Connecting the Tattletale Model 8 to the Computer.....	2-3
Installing the Tattletale Software	2-3
Installing TxTools, TxBASiC and the BYOB onto the Hard Drive.....	2-3
Loading TxBASiC into the EEPROM of the Model 8.....	2-4
Verifying the Operation of the Tattletale Model 8	2-6
How to Operate the Tattletale Model 8 for Initial Checkout.....	2-6
Verifying Tattletale Operation	2-6
Section 3 - Operating the TxTools Program	
Introduction	3-1
What is TxTools and how is it used?.....	3-1
Learning to Use TxTools on the IBM PC (or Compatible)	3-2
Getting Started with TxTools (IBM PC)	3-2
Keyboard Shortcuts for Mouse Actions (IBM PC)	3-4
Explanations of TxTool Menu and Window Options (IBM PC)	3-5
Introduction (IBM PC).....	3-5

Contents (continued)

<u>Title</u>	<u>Page</u>
Explanation of TxTool Window Types (IBM PC)	3-5
Terminal Window (IBM PC)	3-5
Edit File Window Description (IBM PC)	3-7
File Menu Option Descriptions (IBM PC).....	3-8
Edit Menu Option Descriptions (IBM PC).....	3-11
Search Menu Option Descriptions (IBM PC).....	3-12
Tattletale Menu Option Descriptions (IBM PC).....	3-14
CommPort Menu Option Descriptions (IBM PC)	3-18
Windows Menu Option Descriptions (IBM PC)	3-20
Help Menu Option Descriptions (IBM PC)	3-21
Learning to Use TxTools on the Macintosh	3-22
Getting Started with TxTools (Macintosh)	3-22
Keyboard Shortcuts for Mouse Actions (Macintosh)	3-24
Explanations of TxTool Menu and Window Options (Macintosh)	3-25
Introduction (Macintosh)	3-25
Explanation of TxTool Window Types (Macintosh).....	3-25
Terminal Window (Macintosh)	3-25
Edit File Window Description (Macintosh)	3-27
Apple Menu Option Descriptions (Macintosh).....	3-27
File Menu Option Descriptions (Macintosh)	3-28
Edit Menu Option Descriptions (Macintosh).....	3-30
Search Menu Option Descriptions (Macintosh)	3-31
Tattletale Menu Option Descriptions (Macintosh)	3-34
Terminal Menu Option Descriptions (Macintosh).....	3-39
Windows Menu Option Descriptions (Macintosh).....	3-43
Software Change Information for TxTools (IBM PC and Macintosh)	3-44
Program Parameters Saved in the Configuration File (IBM PC Only)	3-44

Section 4 - Using TxBASIC

Introduction	4-1
General TxBASIC Information.....	4-1
TxBASIC Structure.....	4-6
Learning to Use TxBASIC	4-9
Flowcharts.....	4-9
Learning to Build a Data Logger, One Step at a Time	4-10
Tattletale Error Messages	4-24
Advanced use of TxBASIC	4-25
Dual Tasking.....	4-25
TxBASIC Assembly Language.....	4-28
General TxBASIC Assembly Language Information	4-28
Assembly Language Subroutines	4-41
TxBASIC Memory Maps.....	4-44

Contents (continued)

<u>Title</u>	<u>Page</u>
TxBASIC Variables	4-44
Read-Only Variables.....	4-45
TxBASIC Floating Point	4-46
Converting TT BASIC Programs to TxBASIC	4-51
TxBASIC Power-up Program Launch	4-59
Loading your own Programs into the EPROM (Model 2A, 2B, 5 and 6 only)	4-59
Using The ReMinder™ Programmer (Model 2A, 2B, 5 and 6 only)	4-60
Loading your own Programs into the EEPROM (Model 5F, 5F-LCD and 6F only) ...	4-61
Offloading Your Program and the Operating System from RAM	4-61
Erasing the Tattletale EEPROM (Model 5F, 5F-LCD and 6F only)	4-62
Loading your Program into the Erased EEPROM (Model 5F, 5F-LCD and 6F)	4-64
Loading your own Programs into the EEPROM (Model 8 only)	4-64
Possible Problems Erasing an EEPROM (Models 5F, 5F-LCD and 6F only)	4-65
Possible Problems Erasing an EPROM (Model 5 only)	4-65
Installing or Updating the TxBASIC Operating System	4-66
Loading TxBASIC into the EEPROM of the Tattletale (Model 5F, 5F-LCD and 6F)	4-66
Erasing the Tattletale EEPROM (Model 5F, 5F-LCD and 6F)	4-66
Loading TxBASIC into the Erased EEPROM (Model 5F, 5F-LCD and 6F) ...	4-67
Loading TxBASIC into the EEPROM of the Model 8	4-67
How to Completely Erase the Flash EEPROM Memory (Model 8 only)	4-69

Section 4 - Model 8 TxBASIC Addendum

Introduction	4-1
Build Your Own Basic (BYOB)	4-1

Section 5 - TxBASIC Command Reference

How to Use this Section	5-1
TxBASIC Command Quick Reference	5-2
Strings in TxBASIC	5-6
String Functions in TxBASIC.....	5-6
Other String Handling Operations	5-7
Character Constants	5-8

Section 5 - TxBASIC Command Reference Addendum

How to Use this Addendum	5-1
TxBASIC Command Quick Reference	5-1

Section 6 - Hardware and Interface Specifications

Getting Started	6-1
-----------------------	-----

Contents (continued)

<u>Title</u>	<u>Page</u>
Tattletale Model 8 Connectors	6-1
Pin and Socket Connector Specifications	6-1
SquishyBus Connector Specifications	6-2
Mounting the Model 8 to the Prototyping Boards	6-2
Prototyping Board Details	6-3
Battery Power.....	6-3
The UART	6-5
Tattletale Model 8 Connections and Specifications	6-7
Digital I/O Line Connections.....	6-13
Important Information before using the Digital I/O Lines	6-15
Analog Input Connections and Specifications	6-17
12-Bit, 8-Channel A-D Converter	6-17
Analog Inputs	6-17
Main UART	6-18
Changing the Baud Rate	6-18
Power Supply Considerations.....	6-18
Current Drain	6-19
A-D Converter Circuit	6-20
RESET	6-20
Memory.....	6-20
RAM	6-20
Flash Memory	6-20
QSM - Queued Serial Module	6-20
SCI - Serial Communications Interface	6-20
SIM - System Integration Module	6-21
System Clock	6-21
PIC 16C64.....	6-21
Oscillator.....	6-21
Model 8 Accessories	6-22
I/O-8 Prototyping Board	6-22
PR-8 Prototyping Board.....	6-22
Aztec C Compiler for DOS.....	6-23
Aztec C Source Level Remote Debugger	6-23
Tattletale 8 C Libraries for DOS.....	6-23
SquishyBus Connectors	6-23
PCMCIA Card Adapter (Obsolete, contact Onset for alternate part.)	6-24

Section 7 - Application Notes

Adding a 16 x 2 Display to the Tattletale	7-1
Storing Date and Time in Binary	7-2
122 KHz A-D for Tattletale loggers (Model 4A only)	7-3
Adding More A-D channels	7-5

Contents (continued)

<u>Title</u>	<u>Page</u>
12-Bit 10KHz A-D (Model 5 only)	7-6
Other Protocols for the Tattletale UART	7-8
Convert Bipolar Input to Unipolar	7-10
Operational Amplifiers and Instrumentation Amplifiers	7-11
Digital Output Protection	7-15
Digital Input Protection	7-16
Input Protection	7-17
Alternate System Clock	7-19
Overview of the Timing Sub-system of a Tattletale	7-19
Need for an Alternate System Clock	7-19
Using the Alternate System Clock	7-19
Alternate Clock Program (ALTCLOCK.TXB)	7-20
New Baud Rate Program (NEWBAUDS.TXB)	7-22

Section 8 - Troubleshooting

Introduction	8-1
Troubleshooting Tattletale Problems	8-1
What to do if the Operation Test Fails	8-2
Troubleshooting Procedure #1	8-2
Troubleshooting Procedure #2	8-3
Troubleshooting Procedure #3	8-5
Troubleshooting Procedure #4	8-7
Contacting Onset Computer Product Support	8-9
Sending a FAX to Product Support	8-9
Sending E-mail to Product Support over the Internet	8-9
Calling Onset Computer Product Support	8-9

Appendix A - Manufacturer Contact Numbers

Appendix B - Data Sheets

Glossary

Index

List of Illustrations

<u>Figure</u>	<u>Title</u>	<u>Page</u>
Figure 1-1	Block Diagram of the Four Major Sections of a Tattletale using TxBASIC	1-3
Figure 2-1	IO-8 with Test Components Soldered onto it	2-2
Figure 2-2	Schematic of the IO-8 showing the Temperature Test Circuit	2-2
Figure 3-1	Flow Chart of the Development Cycle for a TxBASIC Program	3-1
Figure 3-2	Save File Dialog Box (IBM PC)	3-2
Figure 3-3	Error Window Showing a Parse Error (IBM PC)	3-3
Figure 3-4	No Errors Window (IBM PC)	3-3
Figure 3-5	Terminal Window Display (IBM PC)	3-6
Figure 3-6	Edit File Window (IBM PC)	3-7
Figure 3-7	Typical Window after Opening a TxTools Program File (IBM PC)	3-8
Figure 3-8	File Menu Options (IBM PC)	3-8
Figure 3-9	Open File Dialog Box (IBM PC)	3-10
Figure 3-10	File Name Extension Dialog Box	3-11
Figure 3-11	Edit Menu Options (IBM PC)	3-11
Figure 3-12	Search Menu Options (IBM PC)	3-12
Figure 3-13	Find Option Dialog Box (IBM PC)	3-13
Figure 3-14	Replace Option Dialog Box (IBM PC)	3-14
Figure 3-15	Tattletale Menu Options (IBM PC)	3-14
Figure 3-16	Off-load Datafile Option Dialog Box (IBM PC)	3-16
Figure 3-17	Disk Off-load Option Dialog Box (IBM PC)	3-17
Figure 3-18	Tokenizer Flags Sub-Menu Options (IBM PC)	3-17
Figure 3-19	CommPort Menu Options (IBM PC)	3-18
Figure 3-20	Baud Rate / Protocol Option Dialog Box (IBM PC)	3-20
Figure 3-21	ASCII Transfer Option Dialog Box (IBM PC)	3-20
Figure 3-22	Windows Menu Options (IBM PC)	3-20
Figure 3-23	Help Menu Options (IBM PC)	3-21
Figure 3-24	Save File Dialog Box (Macintosh)	3-23
Figure 3-25	Error Box Showing a Parse Error (Macintosh)	3-23
Figure 3-26	No Errors Box (Macintosh)	3-23
Figure 3-27	Terminal Window Display (Macintosh)	3-26
Figure 3-28	Edit File Window (Macintosh)	3-27
Figure 3-29	Apple Menu Options (Macintosh)	3-27
Figure 3-30	File Menu Options (Macintosh)	3-28
Figure 3-31	Open File Dialog Box (Macintosh)	3-29
Figure 3-32	Edit Menu Options (Macintosh)	3-30
Figure 3-33	Search Menu Options (Macintosh)	3-31
Figure 3-34	Find Option Dialog Box (Macintosh)	3-33
Figure 3-35	Tattletale Menu Options (Macintosh)	3-34
Figure 3-36	TxBASIC Options Sub-Menu (Macintosh)	3-37
Figure 3-37	Off-load Datafile Option Dialog Box (Macintosh)	3-38
Figure 3-38	Terminal Menu Options (Macintosh)	3-39

List of Illustrations (continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
Figure 3-39	Send File ASCII or Send File XMODEM Dialog Box (Macintosh)	3-40
Figure 3-40	Receive File XMODEM Dialog Box (Macintosh)	3-41
Figure 3-41	Terminal Preferences Sub-Menu (Macintosh)	3-41
Figure 3-42	Baud Rate / Protocol Option Dialog Box (Macintosh)	3-42
Figure 3-43	ASCII Transfer Option Dialog Box (Macintosh)	3-43
Figure 3-44	Windows Menu Options (Macintosh)	3-43
Figure 4-1	Flow Chart of the Development Path for a TxBASIC Program	4-6
Figure 4-2	Tattle Program Parts	4-7
Figure 4-3	Tattletale Interpreter	4-7
Figure 4-4	Tattletale Power-up Sequence	4-8
Figure 4-5	Tutorial Program Flowchart #1	4-11
Figure 4-6	Tutorial Program Flowchart #2	4-13
Figure 4-7	Tutorial Program Flowchart #3	4-14
Figure 4-8	Tutorial Program Flowchart #4	4-16
Figure 4-9	How? Error Line Display	4-24
Figure 4-10	One Pass Through an Assembly Routine	4-30
Figure 4-11	TxBASIC Variable / Expression	4-31
Figure 4-12	Pulse Width Measurement	4-32
Figure 4-13	TxBASIC Memory Maps	4-44
Figure 4-14	Where to Insert the Programming Shunt (Model 5F)	4-62
Figure 4-15	Where to Insert the Programming Shunt (Model 5F-LCD)	4-63
Figure 4-16	Where to Insert the Programming Shunt (Model 6F)	4-63
Figure 4-17	Flowchart for Creating your own TxBASIC Commands	4-1
Figure 5-1	TxBASIC Variable Registers	5-16
Figure 5-2	TxBASIC Variable Registers 2	5-19
Figure 5-3	Schematic for Adding more A-D Channels	5-23
Figure 5-4	5F-LCD Characters that can be Displayed	5-32
Figure 5-5	D-A Smoothing Circuit	5-36
Figure 5-6	5F-LCD Characters that can be Displayed	5-85
Figure 5-7	Example of 17 Bits being Shifted	5-86
Figure 5-8	74HC165 and 74HC166 Shift Registers	5-87
Figure 5-9	Timing Lines used by the SDO Command	5-88
Figure 5-10	SDO Command Timing Lines	5-89
Figure 5-11	Timing Chart for a Sleep 100 Command	5-91
Figure 5-12	Timing Chart Showing Sleep Command Waking Early	5-92
Figure 5-13	Thermistor Circuit	5-101
Figure 6-1	Model 8 Dimensions	6-1
Figure 6-2	SquishyBus Dimensions	6-2
Figure 6-3	DC Power Jack w/o Connector Plugged in	6-4

List of Illustrations (continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
Figure 6-4	DC Power Jack with Connector Plugged in	6-4
Figure 6-5	Communication Cables	6-5
Figure 6-6	Schematic of the Model 8	6-6
Figure 6-7	Diagram of A-D Regulator Circuit	6-20
Figure 6-8	I/O-8 Prototyping Board	6-22
Figure 6-9	PR-8 Prototyping Board	6-23
Figure 6-10	Side View of I/O8, TT8 and PCMCIA Adapter Assembly	6-24
Figure 7-1	Adding a 16 x 2 Display to the Tattletale	7-1
Figure 7-2	122KHz A-D Circuit	7-3
Figure 7-3	Circuit Diagram for Adding more A-D Channels	7-5
Figure 7-4	12-Bit 10KHz A-D Circuit for the Model 5	7-6
Figure 7-5	Modifying the UART Registers	7-8
Figure 7-6	Circuit for Converting Bipolar to Unipolar	7-10
Figure 7-7	Another Circuit for Converting Bipolar to Unipolar	7-11
Figure 7-8	Operational Amplifiers	7-11
Figure 7-9	Instrumentation Amplifier Circuit	7-14
Figure 7-10	Digital Output Protection Circuit	7-15
Figure 7-11	Digital Input Protection Circuit	7-16
Figure 7-12	Input Protection with just a Resistor	7-17
Figure 7-13	Input Protection with a Resistor and Capacitor	7-17
Figure 7-14	Input Protection with a Zener Diode	7-18
Figure 8-1	Troubleshooting Flow Chart	8-2
Figure 8-2	Communication Cable Pin Layouts	8-5
Figure 8-3	Testing the Communication Cable	8-6

List of Tables

<u>Table</u>	<u>Title</u>	<u>Page</u>
Table 1-1	Quick Start Sections for Experienced Users	1-1
Table 1-2	Suggested Section Reading Order for New Users	1-1
Table 1-3	Model 8 Specifications	1-2
Table 1-4	Tattletale Block Diagram Section Descriptions	1-3
Table 1-5	Document Conventions	1-4
Table 1-6	Contents of the TxBASiC/TxTools Diskette	1-7
Table 1-7	Model 8 TxBASiC Development Kit Contents (IBM Version)	1-8
Table 1-8	Model 8 Deluxe Development Kit Contents (IBM Version)	1-8
Table 2-1	Model 8 Communication Settings	2-5
Table 3-1	Keyboard Shortcuts for Mouse Actions (IBM PC)	3-4
Table 3-2	Terminal Window Feature Descriptions (IBM PC)	3-6
Table 3-3	Edit File Window Feature Descriptions (IBM PC)	3-7
Table 3-4	File Menu Option Descriptions (IBM PC)	3-9
Table 3-5	Edit Menu Option Descriptions (IBM PC)	3-11
Table 3-6	Search Menu Option Descriptions (IBM PC)	3-13
Table 3-7	Tattletale Menu Option Descriptions (IBM PC)	3-14
Table 3-8	Tokenizer Flags Sub-Menu Option Descriptions (IBM PC)	3-17
Table 3-9	CommPort Menu Option Descriptions (IBM PC)	3-18
Table 3-10	Window Menu Option Descriptions (IBM PC)	3-20
Table 3-11	Help Menu Option Descriptions (IBM PC)	3-21
Table 3-12	Keyboard Shortcuts for Mouse Actions (Macintosh)	3-24
Table 3-13	Terminal Window Feature Descriptions (Macintosh)	3-26
Table 3-14	Apple Menu Option Descriptions (Macintosh)	3-27
Table 3-15	File Menu Option Descriptions (Macintosh)	3-28
Table 3-16	Edit Menu Option Descriptions (Macintosh)	3-30
Table 3-17	Search Menu Option Descriptions (Macintosh)	3-32
Table 3-18	Find Dialog Box Option Descriptions (Macintosh)	3-33
Table 3-19	Tattletale Menu Option Descriptions (Macintosh)	3-35
Table 3-20	TxBASiC Options Sub-Menu Descriptions (Macintosh)	3-37
Table 3-21	Terminal Menu Option Descriptions (Macintosh)	3-39
Table 3-22	Terminal Preferences Sub-Menu Descriptions (Macintosh)	3-41
Table 3-23	Windows Menu Option Descriptions (Macintosh)	3-43
Table 4-1	TxBASiC Arithmetic Operators	4-1
Table 4-2	Relational Operators	4-5
Table 4-3	Tattletale Internal Monitor Commands	4-8
Table 4-4	Flowchart Symbols	4-9
Table 4-5	TxBASiC "HOW" Error Code Listing	4-24
Table 4-6	Assembly Language Addressing Modes	4-35
Table 4-7	TxBASiC Model Command Formats	4-45

List of Tables (continued)

<u>Table</u>	<u>Title</u>	<u>Page</u>
Table 4-8	Floating Point Operation Functions	4-47
Table 5-1	Command Syntax Usage	5-1
Table 5-2	Syntax Usage Key	5-2
Table 5-3	TxBASIC Command Quick Reference	5-2
Table 5-4	TxBASIC String Commands	5-7
Table 5-5	TxBASIC Model Command Formats	5-62
Table 5-6	TxBASIC Command Quick Reference	5-115
Table 5-7	68332 Clock Rates for the 40000 Crystal	5-127
Table 6-1	Model 8 Specifications	6-7
Table 6-2	I/O-8 Pin Functions	6-8
Table 6-3	PR-8 Pin Functions	6-9
Table 6-4	Digital I/O Lines with Special Functions	6-14
Table 6-5	Detailed Descriptions of the Special I/O Line Functions	6-15
Table 6-6	Digital I/O Line Specifications (from Motorola)	6-16
Table 6-7	Analog Input Pin Specifications	6-17
Table 6-8	Current Drain Due to Various Commands	6-19
Table 6-9	Current Drain Due while using Low Power Modes	6-19
Table 7-1	Possible Tattletale Protocol Settings	7-8
Table 7-2	Op Amp Limitation Data	7-12
Table 8-1	Tattletale Communication Default Settings	8-1
Table 8-2	Troubleshooting Pins to Test	8-8

Section 1 - Introduction to the Model 8

Quick Start Information

If you are already an experienced Tattletale user and you want to jump right in and start using the Tattletale, read the sections listed in Table 1-1 for detailed information regarding this specific model of the Tattletale line.

If you are a new user of Tattletale products, read the sections listed in Table 1-2 in the order listed for the easiest learning curve.

Table 1-1: Quick Start Sections for Experienced Users

Section Number	Description of Section Contents
2 - Installation	Step-by-step instructions for connecting the Tattletale to your computer system
6 - Hardware Specifications	Detailed design specifications for this Tattletale model. This information is needed for designing interfaces for the Tattletale
7 - Application Notes	Detailed examples showing interface techniques for many different common uses for the Tattletale

Table 1-2: Suggested Section Reading Order for New Users

Section Number	Description of Section Contents
1 - Introduction	Explains general information and safety information about the Tattletale
2 - Installation	Step-by-step instructions for connecting the Tattletale to your computer system
3 - Operating TxTools	A step -by- step tutorial for learning TxTools and detailed descriptions of all the menu options in TxTools
4 - Using TxBASiC	A step -by- step tutorial for learning TxBASiC and detailed information on using TxBASiC to operate the Tattletale
5 - TxBASiC Commands	Detailed information on using TxBASiC commands to operate the Tattletale. All possible commands are covered in this section
6 - Hardware Specifications	Detailed design specifications for this Tattletale model. This information is needed for designing interfaces for the Tattletale
7 - Application Notes	Detailed examples showing interface techniques for many different common uses for the Tattletale

Welcome to the Tattletale Model 8

Table 1-3: Model 8 Specifications

Size (inches)	2 x 3 x 0.5
Weight (oz.)	1
Processor	68332
Data capacity (RAM)	256K/1M
Additional capacity	PCMCIA
Flash EEPROM	256K
A-D converter	12-bit
Analog channels	8
Max sampling rate (Hz)	100K
Digital I/O lines	up to 25
Count channels	up to 25
Minimum current	<200 μ A typical
Peak current	150mA
Main UART baud (default) at RS-232 Levels:	9600
TPU UART baud rates (others available):	The 14 TPU lines can be set to any standard rate up to 500K
Serial EEPROM (bytes)	7190
Voltage input	7 to 15V
Battery RAM backup	No
Real-time clock	Hardware
Programming languages	C, Tx BASIC
Operating temperature range	-40 to +85 $^{\circ}$ C
Relative humidity range	0 - 95% non-condensing
An RS-232 port is used for communication	

Congratulations on purchasing the Tattletale Model 8. The Model 8 is one of the smallest Tattletalets available yet offers large data storage capacity, low power drain and great flexibility in a multi-channel, non-dedicated logger/controller. The compact design of the Tattletale Model 8 places the most important components on a single 2in. x 3in. x 1/2in. printed circuit board. The Model 8 can be described as having four functional sections, each one described in Table 1-4 and shown Figure 1-1.

Operating Temperature Range

Model 8 components are specified to operate over a temperature range of -40 $^{\circ}$ C to +85 $^{\circ}$ C with the following exceptions. The switch used to enter the Background Debugging Mode (BDM) during the power up sequence has an operating range of -20 $^{\circ}$ C to +70 $^{\circ}$ C. The Light Emitting Diode (LED) used to indicate a low signal on IRQ3 (pin 61 on PR-8, pin A-5 on IO-8) has an operating range of -30 $^{\circ}$ C to +85 $^{\circ}$ C.

Table 1-4: Tattletale Block Diagram Section Descriptions

Section on Block Diagram	Description
A	Analog and digital I/O, including UARTs, individually programmable digital I/O lines, counter, square wave generator and three-wire serial interface.
B	CMOS CPU, CMOS RAM and FLASH EEPROM for non-volatile program storage.
C	Data storage (the Datafile) for storing the results of measurements.
D	Voltage regulator to control supply voltages from a battery input or 7 - 15V power supply.

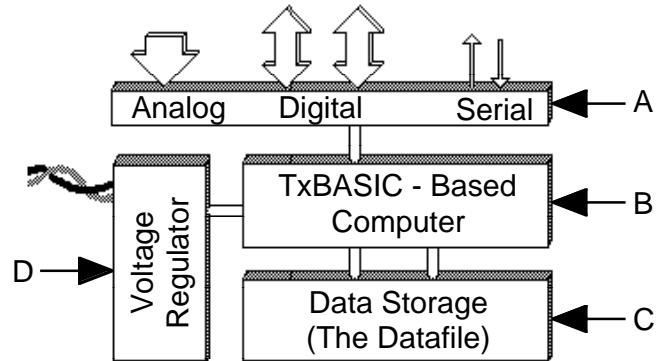


Figure 1-1: Block Diagram of the Four Major Sections of a Tattletale using TxBASiC

The Model 8 has up to 25 digital I/O lines and an 8 channel, 12-bit A-D converter; it can be powered by any 7 to 15V power supply. The Model 8 has two on board voltage regulators: one for the digital circuits and another for the A-D converter. Both regulators are current and thermally limited, protecting them from unintentional overloads during development, and have about 50mA of excess capacity for powering external circuits (when using a power supply with 200mA of current).

Onset's TxBASiC operating system dramatically reduces product development time and costs. We are sure that you will be very pleased with your Tattletale purchase.

Onset Computer Corporation

Onset Computer has specialized in the design and manufacture of low power computers for data logging and control applications since 1981. Our machines fly on the Space Shuttle, monitor conditions at the bottom of the ocean and control a myriad of data gathering systems worldwide.

Onset's StowAway™, Hobo and Tattletale lines have gathered data in the world's

oceans, in balloons, aircraft, parachutes, race cars, boats, trains, pipelines, animals, humans, oil fields, forests and streams.

The company's first product was the C-44 bus card set, designed specifically for battery-powered applications. Building on this concept, Onset engineers developed the Tattletale line of machine control and data logging engines. From the beginning, the Tattletales offered significant improvements, reducing physical size while facilitating program development by compressing the capabilities of a rack of C-44 bus cards onto a single board and adding built-in BASIC. Further developments included the addition of plotting software and increased processing capability. Your Tattletale Model 8 is the product of thirteen years of refinement.

Onset has also developed a line of low cost, single channel, dedicated data loggers. Hobo and StowAway data loggers are configured and launched with BoxCar or LogBook host software. When the logger's mission is complete, the software downloads the collected data and displays it graphically. BoxCar and LogBook also provide data conversion to delimited text formats that can be imported into spreadsheets and word processors. The Hobo/StowAway line features non-volatile data storage, incredibly small size and exceptionally low power drain. BoxCar and LogBook software is available for both the Macintosh and IBM PC environments.

Conventions used in this Document

To help you identify commands, information and safety warnings easily, this manual uses the text formats and visual aids listed in Table 1-5.

Table 1-5: Document Conventions

Type Style	Used for
bold lowercase	Command names for the IBM PC. Any text in this format must be typed exactly as shown or the commands will not work correctly.
<i>italic</i>	Used to emphasize important information in a step or paragraph.
ALL CAPITALS	Directory names and file names on the IBM PC and acronyms. Also the RETURN key and the ENTER key will be in all capitals when you are being told to press them in a procedure.
NOTICE boxes	Indicates that equipment could be damaged if the instructions that follow the notice are not strictly observed.

NOTE: References made to figures, tables and specific pages will always show the Section number first and then the page or reference number.

Where to Start

Having purchased a development kit, you may be surprised (terrified?) by the sheer volume of manuals and documentation. Fortunately, most of this material needs only infrequent reference; and then only when accessing some of the 8's more esoteric capabilities. Everyone should read Section 2. It has the information you need to safely proceed to the next stage—hooking up the Model 8 and getting down to business.

After reading Section 2, you may want to reread or continue reading the remainder of this section, since much of this information will make more sense as you learn more about the Model 8.

What You Should Know

We assume that you already know how to program in BASIC. The combination of TxBASIC and a fairly complex piece of hardware like the Model 8 make it an inappropriate vehicle for learning to program.

Warnings and Precautions

If you've read ahead to Section 2, you are probably feeling a little timorous about even touching the board. If so, we've achieved our goal: the Tattletale has survived the introductory phase. Though we want you to take the warnings to heart, the reality is that the Model 8 is a remarkably robust board, capable of surviving even in a busy development environment.

Additional Information Resources

We try to keep the printed documentation and development kit diskettes up to date, but they invariably lag behind the electronically distributed files and documentation available on our bulletin board system and on internet (see Tech Support Section). If you have access to our BBS or the internet, periodically check for new and interesting files. In addition, your purchase puts you on our mailing list for the TattleTips newsletter in which we announce significant new offerings and provide information listing the latest software revisions. Some files, such as TxBASIC for the Model 8, are password protected with new keys applied to each new release. If you are a registered customer, call Onset at (508) 759-9500 between 9 AM and 5 PM EST with the name of the file you want to access, and we will provide you with the key.

The PC development diskettes have an EXAMPLES directory which contains additional information and examples. Some of these are of a more technical nature which generally are not required for simple data logger applications, but may be very valuable if you are creating more demanding applications. All of the documentation files are distributed as Adobe Acrobat Documents—an encapsulated document display utility that can be viewed with a Macintosh or Windows.

Documentation

The Model 8 is a complex and powerful machine with many different features. As a result, the descriptions of many components are beyond the scope of this manual.

If you're not sure of where to look for information on a particular topic related to the Model 8, below is a summary of where to start:

TT8 Installation and Operation Manual

Model 8 specific information:

Installation
TOM8 Mini-Monitor
TxTools Communications Software
Model 8 TxBASIC Command Libraries
Hardware
Troubleshooting

Motorola Manuals:
68332 Manual
CPU32 Manual
TPU Manual

General information about the various functions and modules of the 68332 System Integration Module.
Specific information about CPU (Instruction Set).
Specific information about the Time Processor Unit.

LTC1121 Data Sheet

Information about Linear Technologies LTC1121, Low Power Voltage Regulator.

LTC1174 Data Sheet

Information about Linear Technologies LTC1174, DC/DC Converter.

MAX186 Data Sheet

Information about the Maxim MAX186, 12 Bit A/D Converter.

MAX242 Data Sheet

Information about the Maxim MAX242, RS-232 Driver.

Technical Support

Onset should be your first line of defense for problems with Model 8 hardware and software although we may vector your cross-development questions to the appropriate vendor.

Since the Model 8 lends itself to the creation of complex programs, if you find what you believe to be a bug, you will have to reduce the complexity of your application to focus on just the failing portion so that we can reproduce and correct the fault. The problem is often found during this process. Also, to continue to provide free technical support for the Model 8, we have to limit help to questions relating directly to the Model 8 and its libraries and ask that you direct general programming questions to local consultants or your own in-house experts.

Onset Phone: (508) 759-9500
Onset Fax: (508) 759-9100
Onset Web: www.onsetcomp.com

Development Software Description

IBM Software

By purchasing either the TxBASiC or the Deluxe Development Kit, you should have received two or three 1.4 megabyte floppy disks (depending on your purchase). Table 1-6 shows the files and directories that the TxBASiC disk contains. The other disks will not be used with TxBASiC and should be saved in case you decide to program the Model 8 in C language.

Table 1-6: Contents of the TxBASiC/TxTools Diskette

File / Directory Name Disk-D-8-TXB	Description
<u>TxTools Directory:</u> TXTOOLS.CFG TXTOOLS.CRS TXTOOLS.EXE TXTOOLS.HLP README.TXT EXAMPLES (DIR)	TxTools configuration file TxTools resource file TxTools executable program TxTools resource file for the help menu Last minute notes about the program and bug fixes Samples of working TxBASiC programs
<u>BYOB Directory:</u> BYOB.LIB CUSTEXT.C CUSTMES.C MAKEFILE README.TXT TXBASiC.H	These files require the C compiler (additional): Used with the deluxe development kit Used with the deluxe development kit Used with the deluxe development kit Used with the deluxe development kit Used with the deluxe development kit Used with the deluxe development kit
<u>TXBASiC Directory:</u> README.TXT TXBASiC.AHX TXBASiC.RHX	Information about the TxBASiC version on the disk TxBASiC image for the Model 8 EEPROM Used with the deluxe development kit

Macintosh Software

The TxTools software is available for the Macintosh for use with the Model 8. If you plan to do any development for the Model 8 in C, you must do the development on the IBM PC (or compatible). You can only use the Macintosh with TxBASiC for the Model 8.

At the time this manual was written, TxTools for the Macintosh was being updated so that the menu options between the Macintosh and the IBM PC version of TxTools would resemble each other. The majority of this manual supports both platforms.

Getting Started

Tattletale Model 8 Development Kit Contents

Table 1-7 shows the kit contents for the IBM PC Tattletale development kits.

Table 1-7: Model 8 TxBASiC Development Kit Contents (IBM Version)

Part Numbers for Kit TT8-TXB-DK-DOS	Description
Finished Goods TT8 IO-8 PC-3.5 Cable	Model 8 Tattletale (Purchased separately) 2 x 3 inch prototyping board for the Tattletale Model 8 Communications cable to connect the Tattletale to a PC
Manuals MAN-TT8TXB MAN-MC68332 MAN-CPU-32 MAN-TPU	Manual for the Tattletale Model 8 for use with TxBASiC 3-Ring binder for the manual Tab set for dividing the sections of the manual 3.5in. looseleaf diskette holder Motorola manual for the 68332 processor Motorola manual for the CPU Motorola manual for the Time Processor Unit
Diskettes DISK-D-8-TXB DISK-D-8-XCT	TxBASiC, Build Your Own Basic and TxTools software CrossCut program software
Misc. Goody Bag	Thermistor, 10K resistor and FET for experimenting

Table 1-8: Model 8 Deluxe Development Kit Contents (IBM Version)

Part Numbers for Kit TT8-DLX-DK-DOS	Description
Finished Goods TT8 IO-8 PR-8 PC-3.5 Cable	Model 8 Tattletale (Purchased separately) 2 x 3 inch prototyping board for the Tattletale Model 8 5 x 7 inch prototyping board for the Tattletale Model 8 Communication cable to connect the Tattletale to a PC
Manuals MAN-TT8TXB MAN-TT8C MAN-MC68332 MAN-CPU-32 MAN-TPU	Manual for the Tattletale Model 8 for use with TxBASiC 3-Ring binder for the manual Tab set for dividing the sections of the manual 3.5in. looseleaf diskette holder Manual sections for using the Model 8 with C programming 3-Ring binder for the manual Tab set for dividing the sections of the manual 3.5in. looseleaf diskette holder Motorola manual for the 68332 processor Motorola manual for the CPU Motorola manual for the Time Processor Unit

Table 1-8: Model 8 Deluxe Development Kit Contents (Continued)(IBM Version)

Part Numbers for Kit TT8-DLX-DK-DOS	Description
Diskettes DISK-D-8-AZC DISK-D-8-TXB DISK-D-8-XCT	Onset C Libraries for using Aztec C with the Model 8 TxBASIC, Build Your Own Basic and TxTools software CrossCut program software
Misc. Goody Bag 1/4in. Nut Driver	Thermistor, 10K resistor and FET for experimenting Nut driver for fastening the Squishy bus to the PR-8

What to do if something is Missing or Broken upon Arrival

If you inspect the contents of your development kit and find that an item is missing or damaged, contact Onset Computer Product Support for assistance at (508) 759-9500.

Tools Required to Connect and Test the Tattletale

NOTE: The “[Verifying the Operation of the Tattletale Model 8](#)” procedure on page 2-6 gives you the option of using an actual temperature sensor by soldering two components (from the goody bag) and a jumper wire to the IO-8 prototyping board. The TxBASIC tutorial also takes advantage of the temperature sensor for a more comprehensive tutorial. By doing so you will have actual sensor data being read by the Tattletale. You may however need to either unsolder the components when done or purchase another IO-8 prototyping board (or make your own) before adding your own circuitry to the Tattletale. You can also leave the temperature sensor on the IO-8 and add your own circuitry to the other A-D channels of the Tattletale.

There are no tools required to connect the Tattletale to your computer; however, to test the battery power, communications cable and channel 7 of the A-D converter of the Tattletale, you will need:

Anti-static wrist strap

Soldering iron

Solder

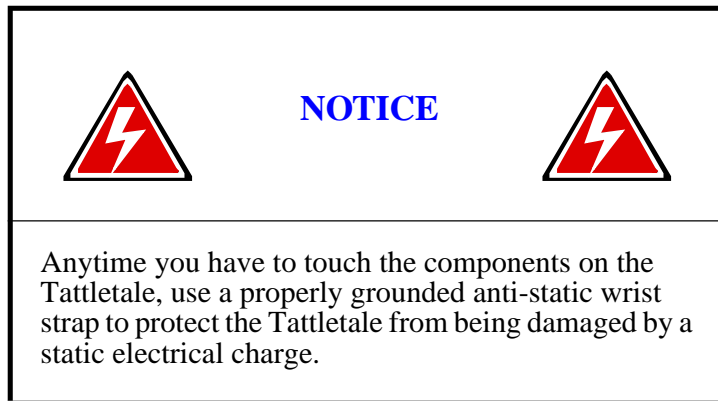
10K thermistor from the goody bag (DALE 9M1002-C3 or YSI 44006)

10K 0.1% resistor from the goody bag (a 5 or 10% resistor can be used if precise results are not needed)

Jumper wire

Safety Precautions

While installing or removing the Tattletale board wear a properly grounded anti-static strap. This will protect the board in the event that your body is holding a static electrical charge which can damage the CMOS chips on the Tattletale.



Proceed to [Section 2 -How to Connect and Setup the Model 8](#) for step-by-step instructions on installing and verifying the operation of the Tattletale.

Section 2 - How to Connect and Setup the Model 8

The Prototyping Board

The prototyping board that comes with the Model 8 development kit allows you to create your own interface for the Tattletale. A prototyping board is required to use the Tattletale Model 8. An I/O-8 prototyping board is included in all the development kits. If you bought the deluxe kit you will also have a PR-8 prototyping board in the development kit (it's the 5 x 7 inch board). We will be using the I/O-8 board for the tutorials. You can design your own interface board using the specifications in [Section 6 - Hardware and Interface Specifications](#). The use of the PR-8 board will be covered in Section - 6.

In order to test the power supply, communications cable and channel 7 of the A-D converter of the Tattletale, it will be necessary to solder a couple of components to the supplied I/O-8 board. After completing the operational testing in this section and the tutorials, you can remove the components from the prototyping board and start designing your own interface. You can also leave the temperature sensor on the IO-8 and add your own circuitry to the other A-D channels of the Tattletale.

Attaching the Tattletale Model 8 to the IO-8 Prototyping Board

Before plugging the Tattletale into the prototyping board, two components from the goody bag and a jumper wire need to be soldered to the board.

Installing a Temporary Sensor onto the Prototyping Board for Testing

Refer to Figure 2-1 and Figure 2-2 for the following steps.

NOTE: If the temperature sensor will be removed after going through the tutorials, you can use a less precise resistor. The only difference is that the temperature data will be less accurate.

1. Solder the 10K thermistor to the pin B11 etch extension and the etch extension for pin B12.

NOTE: Use a heat sink to protect the thermistor while soldering it to the board.

2. Solder a small jumper wire between the pin B12 etch and the etch extension.
3. Solder the 10K 0.1% resistor to the pin B20 etch and the etch extension for pin B12.

NOTE: A 5% or 10% resistor can be used if precise results are not needed.

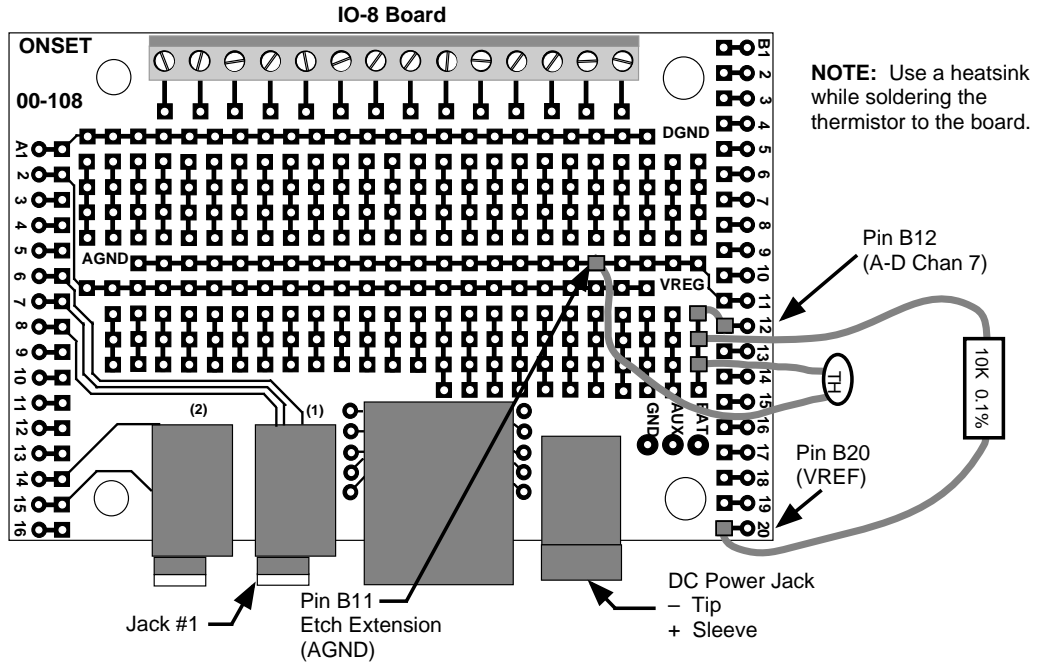


Figure 2-1: IO-8 with Test Components Soldered onto it

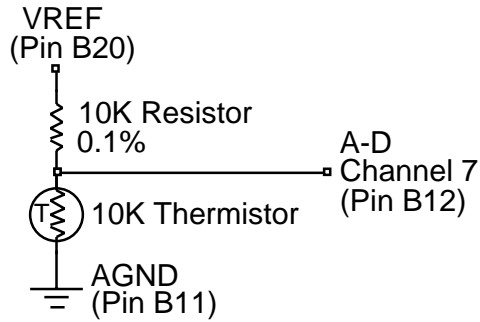
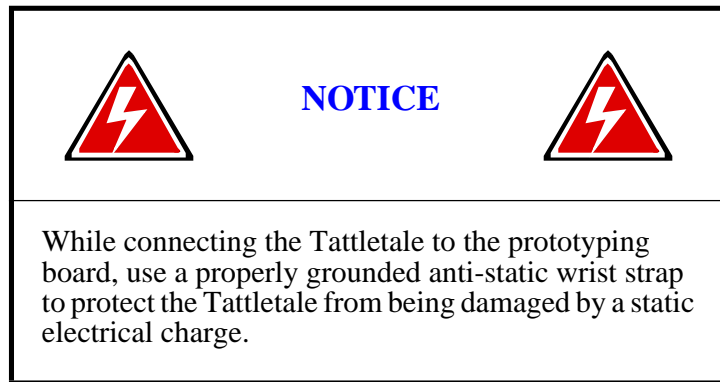


Figure 2-2: Schematic of the IO-8 showing the Temperature Test Circuit

Connecting the Tattletale Model 8 to the Computer



NOTE: DO NOT connect a battery or power supply to the Tattletale until told to do so.

1. Carefully remove the Tattletale from the static protection bag. Touch only the edges of the Tattletale board.
2. Line up the pins of the Tattletale with the sockets of the IO-8 prototyping board (there is a short connector on one end) and carefully mount the Tattletale onto the prototyping board. Make sure that the Tattletale pins are inserted completely into the sockets on the board.
3. Connect the PC-3.5 communication cable to either COM Port 1 or COM Port 2 on the back of your computer. No other port should be used.

Record the serial port to which you have connected the cable. This data is required during the setup of the Tattletale.

4. Insert the other end of the communication cable completely into jack #1 on the IO-8 prototyping board (refer to Figure 2-1).

Installing the Tattletale Software

These files and programs should be installed onto the computer to which the Tattletale was connected.

Installing TxTools, TxBASiC and the BYOB onto the Hard Drive

The files in the TxTools directory will be used extensively while programming in TxBASiC. The TxBASiC directory contains the S-record for permanently loading TxBASiC into the Model 8 and is generally used only once. The BYOB directory (Build Your Own Basic) contains special files that allow you to create new TxBASiC commands; however, these are only usable if you purchased the Aztec C compiler program and will be covered in a separate manual (and is only usable on IBM PC systems).

1. While you are at the DOS prompt on the IBM PC, insert the TxBASiC disk (P/N DISK-D-8-TXB) into the floppy drive.

2. At the DOS C: prompt, enter the command **xcopy a:\ c:\ /-y /v /s** to copy the files from the floppy disk to the hard drive and automatically create new directories named TXTOOLS, TXBASIC and BYOB. (The **-y** causes the program to ask you if you want to overwrite any existing files or directories with identical names. The **/v** verifies the copying process and the **/s** creates and copies any sub-directories.)
3. If you are updating TxTools from an earlier version, please delete all copies of TXTOOLS.CRS, TXTOOLS.CFG and TXTOOLS.HLP (or it may have been called TXHELP.HLP) before loading this new version.

NOTE: This is important to all IBM PC users:

You should place the files TXTOOLS.EXE (the TxTools executable file), TXTOOLS.CFG (TxTools current serial port configuration), TXTOOLS.HLP (the TxTools help file) and TXTOOLS.CRS (a resource file containing the menus and dialog boxes used by TxTools) in a directory that is listed in your PATH, or add the directory they reside in to the PATH statement. This way, no matter where you execute TxTools from, you will always have access to the help system and the resources. Also, it keeps multiple configuration files from being created in each directory you work in.

NOTE: TxTools will not start without access to the resource file TXTOOLS.CRS.

Loading TxBASIC into the EEPROM of the Model 8

The TXBASIC directory contains the S-record file for the TxBASIC language. This S-record is used to burn TxBASIC in the Tattletale EEPROM so that when you power-up, TxBASIC will run automatically. This file is called TXBASIC.AHX.

1. At the DOS prompt, change to the TXTOOLS directory.
2. Enter the command **dir** and make sure that the TXTOOLS.EXE file is in the directory. If it is not, double check that you are in the right directory.
3. Enter the command **txtools -p 1 -b 9600** (If you connected the Tattletale to COM port 2, enter **-p 2** instead of **-p 1**.) The TxTools program will start and open a blank window.

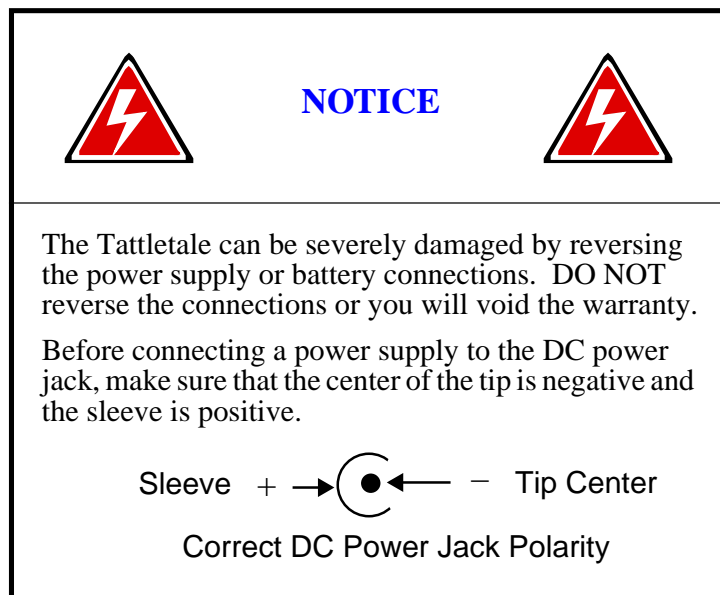
NOTE: The command entered in step 3 automatically sets the com port to the number you specified and the baud rate to the second number you specified. You can enter these manually by selecting “**Port Setup**” from the “**CommPort**” menu. The remaining settings should be left at the default settings listed in Table 2-1. The default settings can be changed as you design your own interface; however, during the installation leave them at the default settings.

Table 2-1: Model 8 Communication Settings

Protocol	Setting
Baud Rate	9600
Data Bits	8
Stop Bits	1
Handshake	None
Parity	None

The default settings can be changed as you design your own interface; however, during the installation leave them at the default settings.

Later when you exit the TxTools program, the settings will be automatically saved for the next session.



Connect a battery or a 7 to 15V power supply to the prototyping board's DC power jack. Make sure that the polarity is not reversed when you connect the power to the board, or you may severely damage the board. We strongly recommend a current limited power source during development. The Model 8 uses up to 150mA of current.

NOTE: DO NOT use a power source that will supply more than 500mA of current or the Model 8 can be severely damaged if a short circuit should occur. We recommend using a 300mA, or less, current limited supply during development for added protection.

4. You should see the "TOM8>" prompt. Pressing the RETURN key should show another TOM8> prompt.
5. From the TOM8> prompt, enter the command "**!o**" and press the RETURN key. A line of text reading: "Waiting for S-Records" will appear.

6. Go to the “**CommPort**” menu at the top of the screen (either by clicking on it with the mouse or using the Alt-C key combination) and select the item - labeled “**Snd file ASCII**”. You will see a small file-finder box open. Find and open the file in the TxBASIC directory by typing “c:\TXBASIC\TXBASIC.AHX” in the “Name” box and press the RETURN key.

NOTE: The 'c' character should be replaced with the designation of the disk drive containing the TxBASIC directory.

A status dialog will appear showing the progress of the load. TxBASIC is quite large and will take about 5 minutes to load. After the file is loaded, you will see a message like:

```
Target is Flash!  
start addr = 00002000  
end addr = 00016181  
Ok to write flash between above addresses? (Y/N)
```

7. Type a 'Y' and you will see a string of asterisks and finally see the TOM8> monitor prompt again. Now you can power-off the Tattletale and power it up again. The following TxBASIC sign-on message will be displayed ending with the TxBASIC prompt:

```
Tattletale Model 8, TxBASIC Version X.XX  
(C) 1998 Onset Computer, Pocasset, MA, USA  
  
TxB#
```

8. Press the RETURN key several times to make sure the TxB# prompt repeats which means everything is okay. You can now proceed with the testing procedures.

Verifying the Operation of the Tattletale Model 8

Before you start interfacing your own products to the Tattletale, test it with the following procedure to make sure that the battery or power supply, communications cable and channel 7 of the A-D converter is functioning correctly.

How to Operate the Tattletale Model 8 for Initial Checkout

Verifying Tattletale Operation

1. Disconnect the power source from the DC power jack for a moment and then reconnect the power source.
2. As soon as the power source is connected, the following should be displayed:

```
Tattletale Model 8, TxBASIC Version X.XX  
(C) 1998 Onset Computer, Pocasset, MA, USA  
  
TxB#
```

3. Press the RETURN or ENTER key. The TxB# symbol should be displayed again, if you get no response from the return or enter key or any other key, refer to the [“What to do if the Operation Test Fails”](#) procedure on page 8-2.

When the TxB# symbol repeats, it proves that the serial interface can send as well as receive. You are now ready to program the Tattletale.

4. Pull down the **“File”** menu and select **“Open”**. The open file dialog box will appear. From the EXAMPLES directory (located in the TxTools directory), open the file named “TEMPTEST.TXB”. A new window will open with “TEMPTEST.TXB” in the title bar and the program will be displayed.
5. We will be explaining the complete operation of the TxTools program in section 4, so for now pull down the **“Tattletale”** menu and select **“Run”**. The program will be loaded into the Tattletale and run.
6. The program will ask you to enter which channel the sensor is connected to. Enter 7 and press the RETURN key.
7. Enter the number of readings you want displayed and press the RETURN key. If you soldered the suggested components onto the IO-8 board correctly, the current temperature in Fahrenheit will be displayed the number of times you entered. This verifies the proper operation of the software, the computer’s serial port, the communications cable, the Tattletale, the battery or power supply and channel 7 of the A-D converter. You can run the program as many times as you want. When you are finished, select **“Quit”** from the TxTools **“File”** menu.

NOTE: If you decide to experiment with some of your own programs now, remember that all Model 8 TxBASIC programs must start with the line “MODEL 800”.

This completes the installation and operational verification of the Tattletale. For additional information on the TxTools software (and a tutorial for TxTools operation) refer to [Section 3 - Operating the TxTools Program](#).

To write your own programs in TxBASIC (and to perform a TxBASIC tutorial) refer to [Section 4 - Using TxBASIC](#) and [Section 5 - TxBASIC Command Reference](#). These sections show the details of all the commands available to operate the Tattletale.

Refer to [Section 6 - Hardware and Interface Specifications](#) if you are ready to start designing products using the Model 8.

If you purchased the PR-8 board and want to start using it immediately (instead of the I/O-8 board), refer to [Section 6 - Hardware and Interface Specifications](#) for instructions on installing the Squishy bus connectors and mounting the Model 8 to the PR-8 board.

Section 3 - Operating the TxTools Program

Introduction

What is TxTools and how is it used?

TxTools running on a host IBM PC or Macintosh works in collaboration with a companion ROM control program running on the Tattletale to form a complete interactive BASIC development system.

TxTools provides a user-friendly, multi-window programming editor for developing and maintaining your BASIC programs. TxTools also provides an integrated tokenizing compiler for generating efficient BASIC code for the Tattletale and a terminal program with a scrolling history buffer for debugging and interacting with your Tattletale programs.

The ROM control program in the Tattletale communicates with the host computer through one of the serial ports to accept and execute tokenized BASIC programs, interact with a user via print and input statements and offload logged data for final analysis.

The development process for creating and debugging a TxBASIC program is shown in Figure 3-1. TxTools is an important part of the development cycle since it allows you to write, debug and tokenize TxBASIC programs with ease. You will notice that the tutorial for TxTools is very short. TxTools was designed for ease of use and the majority of this section will be used for reference only.

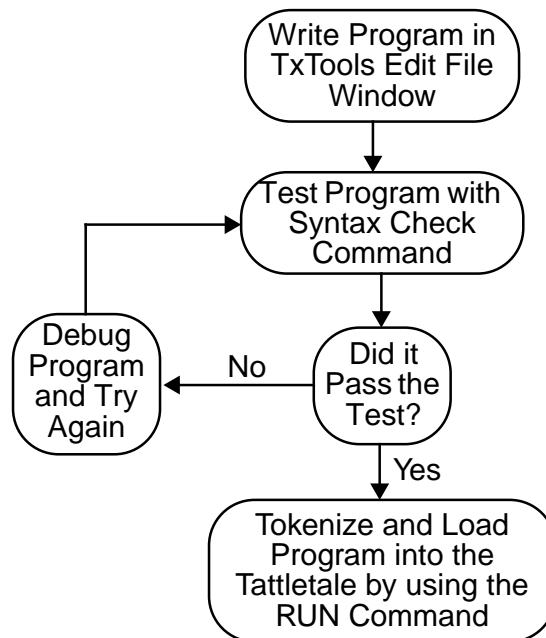


Figure 3-1: Flow Chart of the Development Cycle for a TxBASIC Program

Learning to Use TxTools on the IBM PC (or Compatible)

Getting Started with TxTools (IBM PC)

NOTE: TxTools is only available for DOS, there is not a Windows version but it can run from Windows as a DOS application.

The following procedure shows you step-by-step how to start the TxTools program, open a new editing window and how to get started writing TxBASIC programs. This procedure is specifically for the IBM PC. If you are using a Macintosh, proceed to the [“Learning to Use TxTools on the Macintosh”](#) procedure on page 3-22.

1. From the DOS prompt, go to the directory that has the TxTools executable file in it and enter the command **txtools**. The program will launch and display a blank window (which is called the “Terminal Window” see Figure 3-5 on page 3-6). Refer to the [“Explanations of TxTool Menu and Window Options \(IBM PC\)”](#) on page 3-5 for detailed descriptions of each of the TxTools commands as you perform this procedure.
2. With the communication cable already connected to the Tattletale and to the computer, connect the power supply or battery. The Tattletale startup message will be displayed and the prompt will be a # symbol.
3. Press the ENTER key. The # symbol should be displayed again. This verifies that the serial interface is operating correctly.
4. Pull down the File menu and select New. This will open a new untitled window.

At this point you are ready to start entering a TxBASIC program. For your tutorial of TxTools we will be entering a small program and then debugging it to show you the typical program development path.

5. Type the following exactly as shown including the spaces. (There is an error in the first line on purpose):

```
forx = 1 to 10
print “Hello”
next x
```

6. Pull down the **“File”** menu and select **“Save”**. A dialog box will appear with the cursor in the name box (see Figure 3-2).

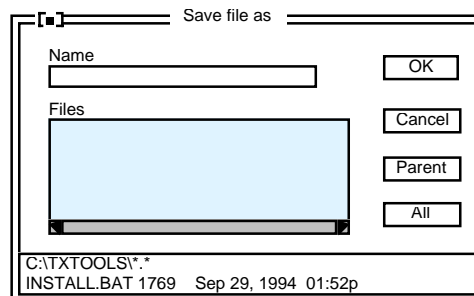


Figure 3-2: Save File Dialog Box (IBM PC)

7. Type the name "TUTORIAL.TXB" and press the ENTER key. The file will be saved in the same directory as the TxTools program.
8. Pull down the Tattletale menu and select "**Syntax Check**" (you can also type ALT-Y). If you typed in the program exactly as was shown, a small error window should have been displayed (see Figure 3-3).



Figure 3-3: Error Window Showing a Parse Error (IBM PC)

9. The error was intentional to show you how to edit and debug your programs. Click on the OK button or press ENTER.
10. Using the mouse click on the "x" after the word "for". The "x" will then be underlined, showing you the current position of the editing cursor. Press the SPACE bar to enter a space between the "x" and the word "for".
11. Enter the command ALT-Y (which is the Syntax Check command). This time Figure 3-4 should be displayed (the numbers displayed will reflect your program).



Figure 3-4: No Errors Window (IBM PC)

The "No Errors" box verifies that the program has no syntax errors. (In technical terms, the Syntax Check command *tokenizes* the program in the edit buffer and reports errors but does not attempt to load the program into the Tattletale.) Press the ENTER key or click on the OK button.

NOTE: Just because the "No Errors" box is displayed does not mean you haven't got any errors in your program, it only means there are no errors in the syntax of the commands used in your program.

12. Enter the command Alt-R (which means RUN) to instruct the Tattletale to run the program currently in RAM (which is our 3 line program right now). The Tattletale will tokenize the program and load it into the Tattletale. It will then switch to the Terminal window and display "Hello" ten times.

Congratulations! - You have written, debugged and run your first TxBASIC program!

This completes the TxTools tutorial. The other TxTools commands and options are explained in detail in the “[Explanations of TxTool Menu and Window Options \(IBM PC\)](#)” on page 3-5. Please proceed to [Section 4 - Using TxBASIC](#) for a tutorial on using TxBASIC and refer to [Section 5 - TxBASIC Command Reference](#) for detailed explanations of each TxBASIC command.

Keyboard Shortcuts for Mouse Actions (IBM PC)

If your IBM PC computer does not have a mouse it will be necessary to use these keyboard commands. Table 3-1 shows all the keyboard commands that are usually operated by using the mouse.

Table 3-1: Keyboard Shortcuts for Mouse Actions (IBM PC)

Menu Option	Command Key	Menu Option	Command Key
Main Menu Selections			
File	Alt-F	CommPort	Alt-C
Edit	Alt-E	Windows	Alt-W
Search	Alt-S	Help	Alt-H
Tattletale	Alt-T		
Sub-Menu Selections			
Open	F3	Run	Alt-R
Save	F2	Load	Alt-L
Quit	Alt-Q	Syntax check	Alt-Y
Cut	Shift-Del	Boot EPROM	Alt-B
Copy	Ctrl-Ins	Offload Datafile	Alt-O
Paste	Shift-Ins	View Variables	Alt-V
Clear	Ctrl-Del	Modify Variables	Alt-M
Paste Date/time	Alt-D	Hex Display	Alt-X
Find Again	Ctrl-L	Capture to File	Alt-Z
Next	F6	Port Setup	Alt-P
Previous	Shift-F6		

Table 3-1: Keyboard Shortcuts for Mouse Actions (IBM PC) (Continued)

Action	Key to Press	Action	Key to Press
Dialog Box			
Cancel	Escape	Toggle check box	Space
OK	Enter	Toggle radio button	Space
Next group	Tab	Next item in group	Up/Down arrow
Editing Controls			
Move cursor	Arrow keys	Move cursor a page	PgUp, PgDn
Move cursor word left	Ctrl-Left arrow	Move cursor word right	Ctrl-Right arrow
Move to line start	Home	Move to line end	End
Delete line	Ctrl-Y	Delete to word end	Ctrl-T
Delete character to left	Backspace	Delete character	Del
Toggle insert mode	Ins	Marking blocks	Shift-Arrow keys
Other Controls			
Send BREAK	Alt-K		
NOTE: If a menu item is “grayed out” it is unavailable in the current mode.			

Explanations of TxTool Menu and Window Options (IBM PC)

Introduction (IBM PC)

This part of the manual describes the window environment and the various areas of the window you need to understand TxTools. A description of each menu and sub-menu option is also included.

Explanation of TxTool Window Types (IBM PC)

Terminal Window (IBM PC)

When you first start TxTools, you will immediately see one window that fills the screen (see Figure 3-5 and Table 3-2). This window displays any characters that are received by the serial port (which is anything sent out from the Tattletale). Keyboard characters go out to the Tattletale while the Terminal window displays any replies being sent from the Tattletale.

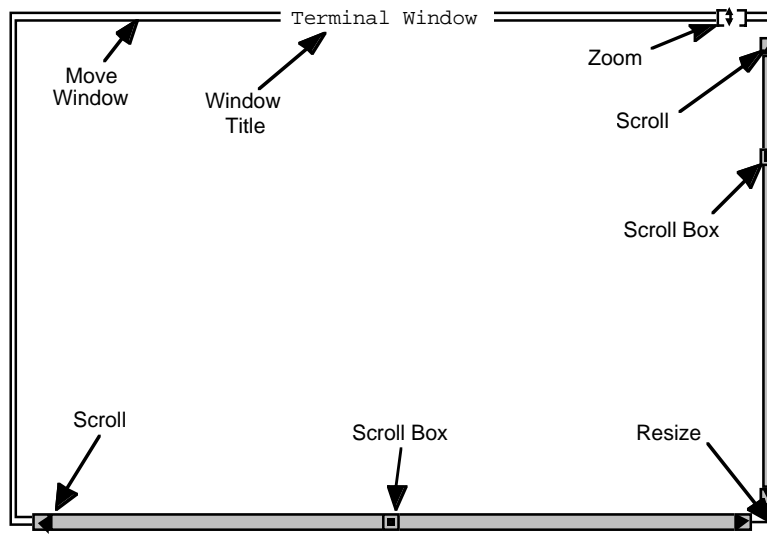


Figure 3-5: Terminal Window Display (IBM PC)

Table 3-2: Terminal Window Feature Descriptions (IBM PC)

Feature	Description
Title	This is always "Terminal Window" for this window but will vary for the Edit file window.
Zoom	By clicking on the Zoom box, you can toggle the size of the window between full screen and a smaller size.
Move	The Move arrow points to the top frame of the window. You can move the window by clicking and holding anywhere along this top frame and then moving the mouse. The window frame will follow the mouse until you release the button.
Scroll Arrows	Scroll arrows cause the text in the window to scroll one line in the direction of the arrow. Click and hold on these arrows to scroll continuously.
Scroll Box	You can move more quickly through a document by click-hold-dragging on the Scroll Box. Moving the Scroll Box toward the top moves closer to the start of the document and moving the Scroll Box toward the bottom moves closer to the end of the document. The window does not scroll until the Scroll Box is released.
Resize	The size of the window can be adjusted by click-hold-dragging on the Resize corner of the window frame.
<p>NOTE: The Terminal window has a 16000 character circular buffer. This holds about 8 pages of packed text. When the buffer fills, the oldest characters are overwritten. The scrolling is adjusted to keep the characters in order and you cannot scroll back too far.</p>	

Edit File Window Description (IBM PC)

When you open a program file or select “**New**” from the “**File**” menu, an Edit File window is displayed (see Figure 3-6, Figure 3-7 and Table 3-3).

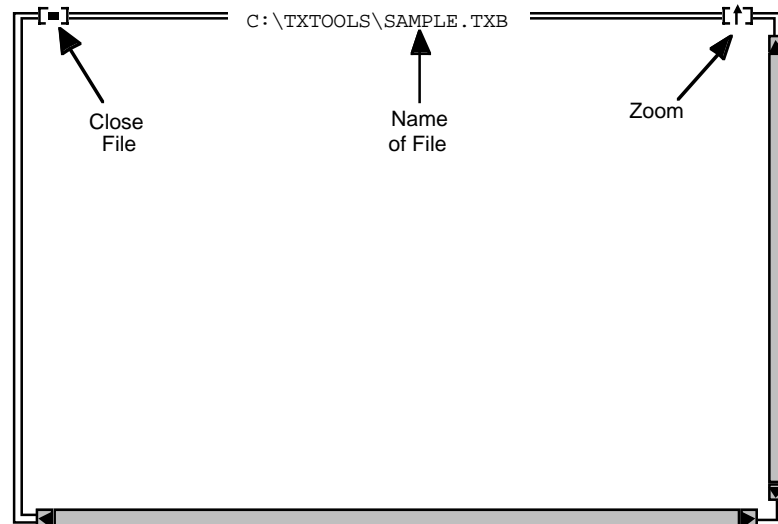


Figure 3-6: Edit File Window (IBM PC)

Table 3-3: Edit File Window Feature Descriptions (IBM PC)

Feature	Description
Close File	The Close box is in the upper left corner (called Cancel in a dialog window). The Close box attempts to close the DOS file displayed in this window. You will be asked if you want to save the changes if the window text has been modified since the last save. You can agree to save changes, throw out all changes since the last save or cancel the close.
Name of File	Displays the full path name of the DOS file.
Zoom	By clicking on the Zoom box, you can toggle the size of the window between full screen and a smaller size.
<p>NOTE: Characters are entered just before the underline “_” cursor. Backspace removes characters before the cursor and Delete removes characters at the cursor. If a block cursor is displayed, the editor is in overwrite mode and each new character typed will overwrite the one under the cursor. The cursor type and editing mode are selected with the INS and DEL keys.</p>	

```

File Edit Search Tattletale CommPort Windows Help
C:\TXTOOLS\TEMPTEST.TXB
for x = 5 to 1 Step -1
Print "Reading Number " ;:print x
sleep 100
let tempValue = temp(chan(0))
tempValue = tempValue * 9 / 5 + 3200
print "The temperature being read by your Tattletale is ";
print tempValue / 100, '.', #02, tempValue % 100;
Print " Degrees F"
Print
next x
print "I'm finished!"
F2 Save AltR Run AltL Load Alt Y Syntax AltO Offld Alt P Port

```

Figure 3-7: Typical Window after Opening a TxTools Program File (IBM PC)

To select a block of text, click and hold the mouse button over the beginning of the block and drag the mouse to the end of the block. When you release the mouse button, the block will be selected. You can then cut, copy, paste or clear this block. Moving the cursor causes the selection to be lost, however. This editor also has a limited Undo capability.

NOTE: As soon as you move the cursor, the Undo buffer is cleared.

The windows continue to exist and be displayed even if they are not in the front (active) window. A window is in the background if it has no close box nor a zoom box.

File Menu Option Descriptions (IBM PC)

NOTE: Typing the underlined character while the menu is open (pulled down) will execute the command (this applies to all the IBM PC menus).

The File menu contains 10 sub-selections (see Figure 3-8 and Table 3-4).

File	Edit	Search	Tattletale	CommPort	Windows	Help
New						
Open...		F3				
Close						
Save		F2				
Save as...						
Print						
Print selection						
Change dir...						
DOS shell						
Quit		Alt-Q				

Figure 3-8: File Menu Options (IBM PC)

Table 3-4: File Menu Option Descriptions (IBM PC)

File Menu Option (Alt-F)	Description
<u>N</u> ew	Opens a new "Untitled" window to enter your program into.
<u>O</u> pen (F3)	Opens an existing program file (see Figure 3-9). When you open a program file, the text of the file is viewed in the Edit File window (Figure 3-6). When you select the open option, you are presented with the open file dialog box (see Figure 3-9).
<u>C</u> lose	Closes the currently active program file. If changes have been made in the file since it was opened, you will be asked if you want to save changes before closing.
<u>S</u> ave (F2)	Saves the program file in the active window to disk. If this is a new file, you will be prompted for a file name to save your program as.
<u>S</u> ave As	Allows you to save the program in the active window to disk under a new name. This is useful when you wish to experiment with your program, but you don't want to make the changes permanent to your original file. If the file already exists, it will ask to confirm before overwriting the file.
<u>P</u> rint	Sends the program in the screen buffer to the DOS PRN device for printing. No header or page eject commands are sent to the printer.
P <u>r</u> int selection	Sends the currently selected text to the printer. If no text is currently selected, you will be notified and no action will take place.
<u>C</u> hange dir	Allows you to change the current DOS working directory. You will be in this new directory when you exit TxTools.
<u>D</u> OS shell	<p>Suspends TxTools and launches a new copy of the DOS shell. This also causes the serial port to be closed. You can execute any DOS commands here even other communications programs. When you're done, use the DOS command EXIT to return to TxTools and the serial port will be opened in the same state it was in before (any characters that were sent to the serial port while running the second DOS session will be lost).</p> <p>NOTE: TxTools is still in memory so there will be limited memory for other programs.</p>
<u>Q</u> uit (Alt-Q)	Exits TxTools, frees up its memory and returns to the DOS command interpreter.

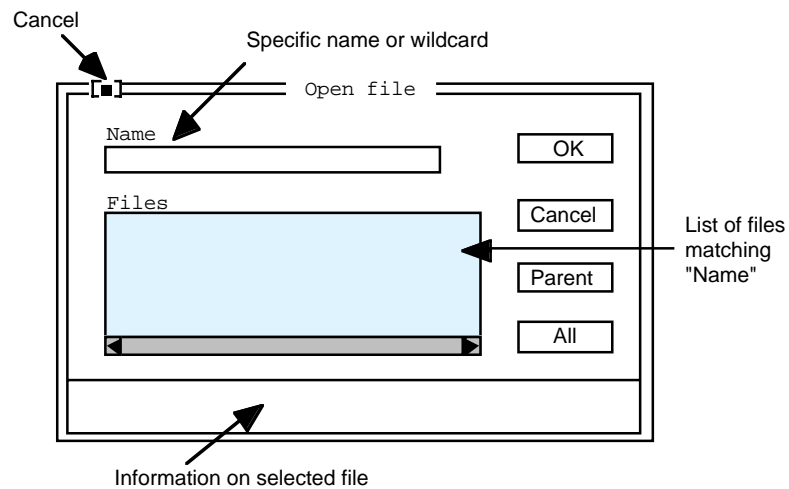


Figure 3-9: Open File Dialog Box (IBM PC)

To Cancel this dialog, you have three choices: click on the Cancel button (under the OK button), click on the Cancel box or press the Escape key.

By default, the Name box will contain *.* and display all files and subdirectories in the current directory. You can enter a wild card string (with optional path) here and the Files box will attempt to show all files matching this string.

NOTE: If you type in the name of a file that doesn't exist, a new window will open with that name. Then if you save the window, a file of that name will be created on your DOS disk. There is always a ".." selection (parent directory) in the Files box (but it is easier to use the parent button). There is information about the currently selected file at the bottom of the dialog. To choose a file for opening, click OK (or press the ENTER key) when the file you want is listed in the "Name" field or double click the file name in the "Files" field.

The bottom button in the "Open" file dialog box can be set for up to five extensions of your own (in addition to the "All" and "Dir" options which are always available). Figure 3-10 shows the file name extension dialog box if you held down the shift key while clicking the "All" button. The bottom button is called a rotating button and will cycle through several options when clicked (the default settings are "All" and "Dir"). To add your own extensions, hold down the Shift key while you click the button labeled "All" and enter up to three characters in each box. These extensions are saved in the CFG file when you exit TxTools. The next time you want to use one of the new extensions, just click on the "All" button several times until the desired extension is displayed.

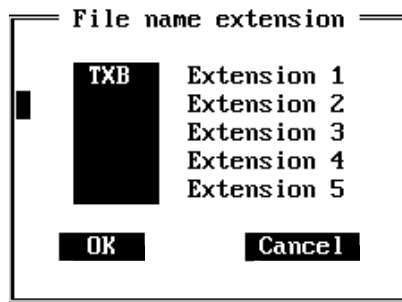


Figure 3-10: File Name Extension Dialog Box

Edit Menu Option Descriptions (IBM PC)

The Edit menu contains 7 sub-selections (see Figure 3-11 and Table 3-5).

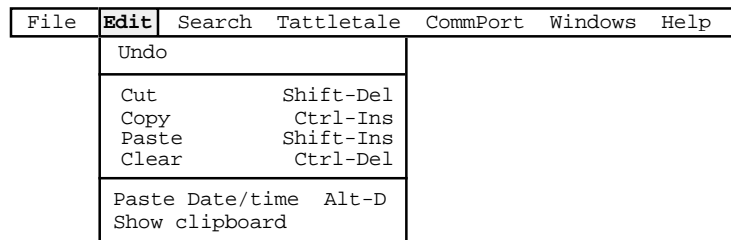


Figure 3-11: Edit Menu Options (IBM PC)

Table 3-5: Edit Menu Option Descriptions (IBM PC)

Edit Menu Option (Alt-E)	Description
<u>U</u> ndo	Select this item to undo an editing action. NOTE: Once you move the cursor, all Undo information is lost.
<u>C</u> ut (Shift-Del)	Remove the currently selected text and save it on the clipboard for pasting later.
<u>C</u> opy (Ctrl-Ins)	Save a copy of the currently selected text on the clipboard for pasting later. Unlike Cut, this does not remove the selected text.
<u>P</u> aste (Shift-Ins)	Insert whatever is on the clipboard at the cursor. Use Cut or Copy to get text into the clipboard.
<u>C</u> lear (Ctrl-Del)	Remove the currently selected text without saving a copy in the clipboard. You can Undo a Clear but not after the cursor has been moved.

Table 3-5: Edit Menu Option Descriptions (IBM PC) (Continued)

Edit Menu Option (Alt-E)	Description
Paste <u>D</u> ate/time (Alt-D)	Inserts a text string showing the current date and time of your PC's clock into the document. If the Terminal window is currently selected, a date and time string will be sent out the serial port to the Tattletale. Selecting this item with no other keys pressed, sends a string of the form: 02/13/94 03:53:52 where 02 is the month, 13 is the day and 94 is the year. If you hold the Shift key down while this is selected, a longer date/time string of this form is used: Friday, February 12, 1994, 03:53 AM. Holding down the Control key while executing this command causes the country-code information in your configuration file to be checked. The date and time will then be pasted in the format normally used in your country (only if you had previously set this in the configuration file).
S <u>h</u> ow clipboard	Opens an editor-type window showing the contents of the Clipboard. You can edit the contents of the Clipboard using the normal editing commands. Only portions of the Clipboard that are selected are available for pasting into other edit windows, so be sure to select that portion of the Clipboard text before exiting this window.

Search Menu Option Descriptions (IBM PC)

The Search menu contains 3 sub-selections (see Figure 3-12 and Table 3-6).

File	Edit	Search	Tattletale	CommPort	Windows	Help
Find... Find Again Ctrl-L Replace...						

Figure 3-12: Search Menu Options (IBM PC)

Table 3-6: Search Menu Option Descriptions (IBM PC)

Search Menu Option (Alt-S)	Description
Find	<p>A dialog box will appear (Figure 3-13) allowing you to search the document for a specific text string. Type the text string to search for in the box labeled "Text to find". You can check either or both of the options "Case sensitive" and "Whole words only" by clicking the mouse between the brackets to the left of the labels or by using the TAB and Arrow keys to work the highlight down to the selection and pressing the SPACE key to toggle the check on and off. Choose "OK" if all selections are correct or "Cancel" to forget the operation and close the dialog box.</p>
Find Again (Ctrl-L)	<p>Once a string has been found with Find, you can continue to look through the document for more occurrences of the same string. It is usually easier to use the Ctrl-L keyboard equivalent for this command.</p>
Replace	<p>A dialog box will appear (Figure 3-14) allowing a text string to be found and replaced with another string. Use of this dialog is similar to that of the Find dialog except that a second text box is available and there are more options.</p> <p>NOTE: Entering nothing in the "New text" box means that found text will be erased.</p> <p>The "Prompt on replace" is normally on (selected with an x). Checking "Prompt on replace" and "Replace all" will automatically look for the next occurrence of "Text to find" after each replace but will query you before replacing text.</p>

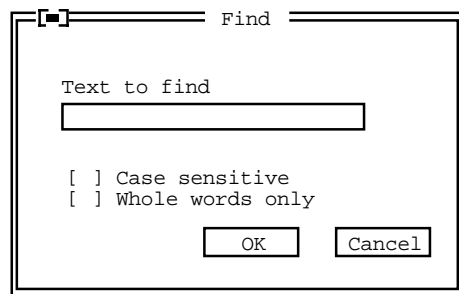


Figure 3-13: Find Option Dialog Box (IBM PC)

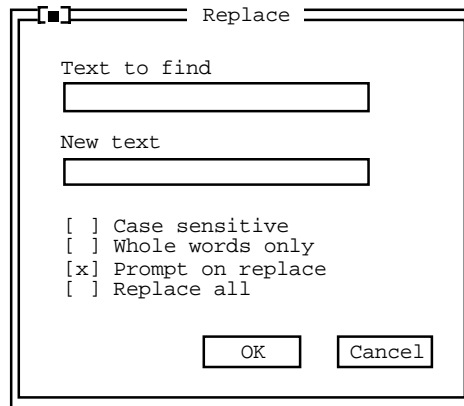


Figure 3-14: Replace Option Dialog Box (IBM PC)

Tattletale Menu Option Descriptions (IBM PC)

The Tattletale menu option contains 10 sub-selections (see Figure 3-15 and Table 3-7).

File	Edit	Search	Tattletale	CommPort	Windows	Help
			Run			Alt-R
			Load			Alt-L
			Syntax check			Alt-Y
			Boot EPROM			Alt-B
			Offload datafile			Alt-O
			Doffload disk			
			Remind EPROM			Alt-E
			View Variables			Alt-V
			Modify Variables			Alt-M
			Options...			

Figure 3-15: Tattletale Menu Options (IBM PC)

Table 3-7: Tattletale Menu Option Descriptions (IBM PC)

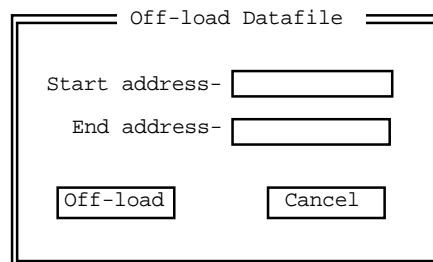
Tattletale Menu Option (Alt-T)	Description
<u>R</u> un (Alt-R)	If selected from the Terminal window, a RUN command is sent to the Tattletale to execute the program currently in RAM. If selected from an edit window, it is tokenized and loaded into the Tattletale. Then the Terminal window is made the active window and a RUN command is sent to the Tattletale.
<u>L</u> oad (Alt-L)	Causes the program in the edit buffer to be tokenized and loaded into the Tattletale. See Table 3-8 for an explanation of the tokenizer flags.

Table 3-7: Tattletale Menu Option Descriptions (IBM PC) (Continued)

Tattletale Menu Option (Alt-T)	Description
Syntax check (Alt-Y)	Tokenizes the program in the edit buffer and reports errors but does not attempt to load the program into the Tattletale. This is affected by the Option flags (see Figure 3-18 and Table 3-8). Upon a successful check, an information box will appear (see Figure 3-4 on page 3-3) showing the size of the tokenized program in bytes, the size of the program header (this is information generated by the tokenizer and will be added to the front of the program) and the size of the variables area used by this program. In the case of a Model 4A or Model 5, a fourth line will show the size of the on-board datafile.
<u>B</u> oot EPROM (Alt-B)	Loads the program burned into the EPROM into RAM and executes it. This also sets up the Tattletale to execute the EPROM program automatically the next time the Tattletale is reset. When you choose this item with the edit window selected, TxTools automatically switches to the Terminal window before booting the EPROM. This works exactly the same for the Model 5F, Model 5F-LCD and the Model 6F all of which use an EEPROM.
<u>O</u> ffload Datafile (Alt-O)	Presents you with a dialog box (see Figure 3-16) allowing you to choose how to off-load the datafile of the Tattletale. To off-load part of the datafile, be sure to fill out the "Start address" and "End address" boxes and click the "Off-load" button (or press the ENTER key on the keyboard). NOTE: For the full datafile, either fill in the appropriate addresses or just enter 0 for both addresses and click the "Off-load" button. After finishing this dialog box, a second dialog is used to select the name of the file. You can save the file anywhere in the DOS directory tree.
<u>D</u> offload Disk Models 6, 6-1M and 6F only	This is similar to the Offload option. It allows off-loading groups of datafiles from the hard disk over the serial port and storing them in one DOS file on your computer. By using the appropriate parallel port card in the computer, the Model 6F can do a fast disk off-load (50K bytes per second) without affecting the RAM datafile using this command. There is a dialog box similar to offload but instead of start and end "datafile addresses" you fill in the start and end "datafile number" (see Figure 3-17). Unlike the Offload Datafile command, you cannot off-load the entire disk by using 0 for both the start and end value, you must fill in the range of the datafiles you want.

Table 3-7: Tattletale Menu Option Descriptions (IBM PC) (Continued)

Tattletale Menu Option (Alt-T)	Description
Remind EPROM	This will create a Hex file containing both your RAM program and all of TxBASIC, sending it through the serial port in XMODEM format. After selecting this, you will be presented with a dialog to select the name and location of the Hex file on your DOS disk.
<u>V</u> iew Variables (Alt-V) <u>M</u> odify Variables (Alt-M)	These commands are enabled only in the Terminal window. You can view or modify variables on the Tattletale without writing a program. When one of these commands is selected, a File Open dialog box appears asking for a Symbol Table name; a symbol table file created by tokenizing a TxBASIC program. A scrollable list of variables will appear. In the View dialog, either double-click on the variable name or use the arrow keys to select a name and press the Enter key. To Modify a variable, you must insert a value in the box at the bottom of the dialog, then, any selected variable will be set to this value. When you tokenize and load a program, a CRC is computed of the variables and their addresses. This CRC is embedded in the Tattletale and stored in the symbol table for that program. When you attempt to use the symbol table to view or modify the variables on the Tattletale, this CRC is compared with the CRC in the Tattletale. If they don't match, you're warned about this. You may be using the symbol table for a program that isn't the one loaded on the Tattletale.
<u>O</u> ptions	Allows flags to be set that affect the tokenizer, the loading process, the off-loading process and the Remind process. See Figure 3-18 and Table 3-8 for a description of all the options.



The dialog box is titled "Off-load Datafile". It contains two input fields: "Start address-" and "End address-". Below these fields are two buttons: "Off-load" and "Cancel".

Figure 3-16: Off-load Datafile Option Dialog Box (IBM PC)

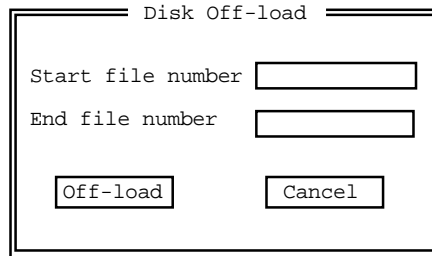


Figure 3-17: Disk Off-load Option Dialog Box (IBM PC)

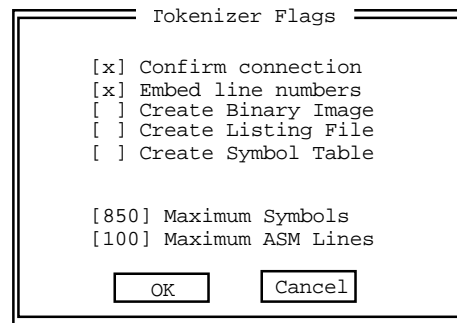


Figure 3-18: Tokenizer Flags Sub-Menu Options (IBM PC)

Table 3-8: Tokenizer Flags Sub-Menu Option Descriptions (IBM PC)

Tokenizer Flags Sub-Menu Options	Description
Confirm Connection	Waits for a reply from the Tattletale before starting a program load, a datafile off-load or a Remind EPROM command.
Embed Line Numbers	Causes the tokenizer to insert a line number in the object program for each line in the source program. This can help in later debugging but adds three bytes per line to the size of the program.
Create Binary Image	The next time the tokenizer runs (whether through the Load command or the Syntax check command) an exact copy of the program (as it's loaded into the Tattletale) is stored in a DOS binary file. The file will have the extension BIN.
Create Listing File	The next time the tokenizer runs (whether through the Load command or the Syntax check command) an annotated listing of the program is stored in a DOS text file. The file will have the extension LST. This can help with debugging a program.

Table 3-8: Tokenizer Flags Sub-Menu Option Descriptions (IBM PC) (Continued)

Tokenizer Flags Sub-Menu Options	Description
Create Symbol Table	The next time the Syntax check command is run, a symbol table will be saved on your DOS disk with extension SYM. A symbol table is always created when you select the “ Load ” command. This can help with debugging a program. It can also help with checking for consistent spelling of all your variables.
Maximum Symbols	Sets the size of the symbol table.
Maximum ASM Lines	Sets the number of assembly code lines you can embed in your program (saving RAM in the precious 640K area).

CommPort Menu Option Descriptions (IBM PC)

The CommPort menu option contains 8 sub-selections (see Figure 3-19 and Table 3-9).

File	Edit	Search	Tattletale	CommPort	Windows	Help
				Snd file ASCII...		
				Rcv file ASCII...		
				Snd file XMODEM...		
				Rcv file XMODEM...		
				Hex display	Alt-X	
				Capture to file...	Alt-Z	
				Port setup...	Alt-P	
				Options...		

Figure 3-19: CommPort Menu Options (IBM PC)

Table 3-9: CommPort Menu Option Descriptions (IBM PC)

CommPort Menu Options (Alt-C)	Description
<u>S</u> nd file ASCII	Allows you to send a DOS text file out the serial port to the Tattletale.
<u>R</u> cv file ASCII	Takes what ever comes in the serial port and stores it in a DOS text file until either an End-of-File character (Ctrl-Z) is received or the Cancel button is pressed.
Snd file <u>X</u> MODEM	Allows you to send a DOS text or binary file out the serial port to the Tattletale using the XMODEM protocol.

Table 3-9: CommPort Menu Option Descriptions (IBM PC) (Continued)

CommPort Menu Options (Alt-C)	Description
Rcv file <u>X</u> MODEM	Starts the XMODEM receive process for a text or binary file coming into the serial port.
<u>H</u> ex display (Alt-X)	Toggles the Terminal window into and out of hexadecimal display mode. In this mode, any incoming characters are displayed in hexadecimal form in rows of 16 characters on the left side of the screen. The ASCII (printable) equivalent is displayed in 16 character rows on the right side of the screen. Toggling this mode always forces the new mode to start on a new line.
Capture to <u>F</u> ile (Alt-Z)	<p>Allows you to toggle the capture mode on and off. If toggling on, you will be presented with a file selection box to choose a DOS file in which to save all input through the comm port (a default name of CAPTURE.TXT is suggested and can be used by simply pressing the ENTER key). Also, the string "CAPTURE" is written to the lower-right corner of the display. In color mode, the status line background is changed to green as another reminder. When you toggle capture off, the file is closed and the status line returned to normal. The default mode is for overwriting an existing file. You can change this to append by holding down the Shift key while selecting Capture mode. You will be asked if you want to change the Capture mode to append.</p> <p>NOTE: To capture to a printer, use the special DOS file name PRN.</p>
<u>P</u> ort setup (Alt-P)	Allows you to set the comm port parameters as shown in Figure 3-20. These values will be stored in a file called TXTOOLS.CFG when you exit TxTools. Notice that these items are groups of radio buttons and only one item of a group can be selected at any one time.
<u>O</u> ptions	The only items implemented here are 'Char delay' and 'Line delay' (see Figure 3-21). They only affect ASCII file transfers and program loads. If Char delay is non-zero, it adds this many milliseconds between characters to both ASCII file transfers and program loads. If Line delay is non-zero, it adds this many milliseconds after sending a carriage return for ASCII file transfers only.

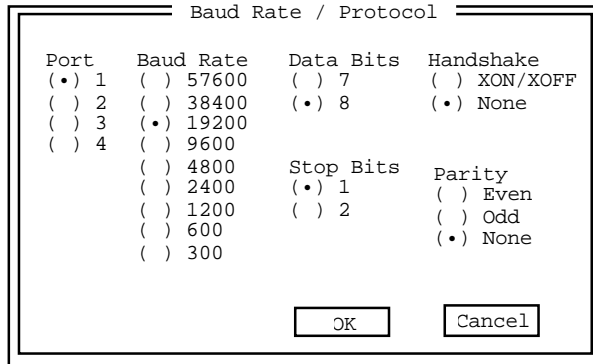


Figure 3-20: Baud Rate / Protocol Option Dialog Box (IBM PC)

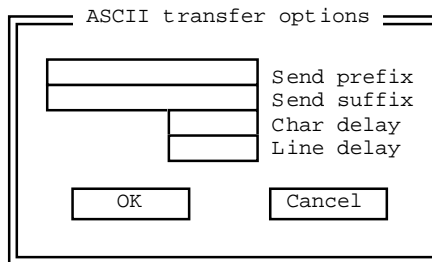


Figure 3-21: ASCII Transfer Option Dialog Box (IBM PC)

Windows Menu Option Descriptions (IBM PC)

This item contains 7 sub-selections (see Figure 3-22 and Table 3-10).

File	Edit	Search	Tattletale	CommPort	Windows	Help
						Tile
						Cascade
						Next
						Previous
						F6
						Shift-F6
						Screen resolution
						Color screen
						Blk/wht screen

Figure 3-22: Windows Menu Options (IBM PC)

Table 3-10: Window Menu Option Descriptions (IBM PC)

Window Menu Options (Alt-W)	Description
<u>T</u> ile and <u>C</u> ascade	These allow you to automatically rearrange all the open windows on your screen.
<u>N</u> ext (F6)	These will make different windows the active window in forward or backward order. These are only needed if a particular window is not visible (in which case, the mouse can be used to make it the active window by clicking on it) or if you have no mouse and need to switch windows.
<u>P</u> revious (Shift-F6)	

Table 3-10: Window Menu Option Descriptions (IBM PC) (Continued)

Window Menu Options (Alt-W)	Description
<u>S</u> creen resolution	If you have an EGA or VGA screen, this command toggles the screen into and out of a higher resolution mode. EGA is capable of a 43 line per screen mode and VGA is capable of a 50 line per screen mode. This has no effect on CGA screens.
<u>C</u> olor screen	Puts a color-capable screen into color mode. Unnecessary if you have a monochrome screen.
<u>B</u> lk/wht screen	Puts the screen in a black and white mode. This is very useful on LCD screens.

Help Menu Option Descriptions (IBM PC)

This item contains 4 sub-selections (see Figure 3-23 and Table 3-11).

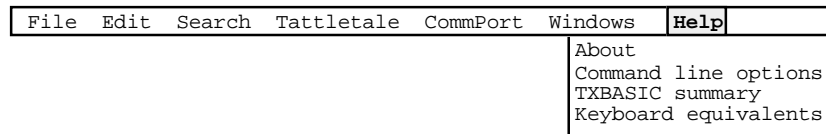


Figure 3-23: Help Menu Options (IBM PC)

Table 3-11: Help Menu Option Descriptions (IBM PC)

Help Menu Options (Alt-H)	Description
<u>A</u> bout	Displays a dialog showing the version number of TxTools you are using.
<u>C</u> ommand line options	Shows options you can set when you start TxTools from the DOS command line.
<u>T</u> xBASIC summary	Displays a list of TxBASIC commands and keywords in alphabetical order. To see more of the list, use the arrow keys or use the mouse to move the scroll box. You can get a brief explanation of a command by double clicking on it with the mouse. Keyboard users can step through the commands with the TAB key and select a command by pressing ENTER. If there is a "See also..." highlighted selection, you can change to that topic by selecting it in the same way you selected the original command. When you're done with the help system, either click on the Close box or press the ESCAPE key.
<u>K</u> eyboard equivalents	A list of TxTools menus and editor actions and the key or key combinations that trigger them. It is essentially a copy of the next section. When you're done with the help system, either click on the Close box or press the ESCAPE key.

Learning to Use TxTools on the Macintosh

Getting Started with TxTools (Macintosh)

The following procedure shows you step-by-step how to start the TxTools program, open a new editing window and how to get started writing TxBASIC programs. This procedure is specifically for the Macintosh, if you are using an IBM PC, proceed to the [“Learning to Use TxTools on the IBM PC \(or Compatible\)”](#) procedure on page 3-2.

1. Go to the folder that has the TxTools application in it and double click on the TxTools application. The program will launch and display a blank window (which is called the “Terminal window” see Figure 3-27 on page 3-26). Refer to the [“Explanations of TxTool Menu and Window Options \(Macintosh\)”](#) on page 3-25 for detailed descriptions of each of the TxTools commands as you perform this procedure.
2. With the communication cable already connected to the Tattletale and to the computer, connect the power supply or battery. The Tattletale startup message will be displayed and the prompt will be a # symbol.
3. Press the RETURN key. The # symbol should be displayed again. This verifies that the serial interface is operating correctly.

NOTE: Pressing the ENTER key on the numeric keypad will result in a What? error.

4. Pull down the File menu and select New. This will open a new untitled window.

At this point you are ready to start entering a TxBASIC program. For your tutorial of TxTools we will be entering a small program and then debugging it to show you the typical program development path.

5. Type the following, exactly as shown, including the spaces. (There is an error in the first line on purpose):

```
forx = 1 to 10
print “Hello”
next x
```

6. Pull down the **“File”** menu and select **“Save”**. A dialog box will appear with the cursor in the name field (see Figure 3-24).

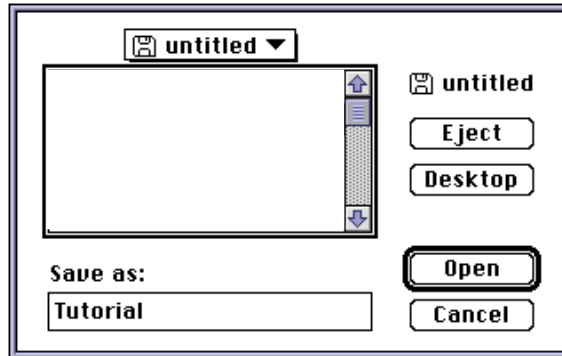


Figure 3-24: Save File Dialog Box (Macintosh)

7. Type the name “Tutorial” and press the RETURN key. The file will be saved in the same folder as the TxTools application.
8. Pull down the Tattletale menu and select “**Syntax Check**” (you can also type Command-Y). If you typed in the program exactly as was shown, a small error window should have been displayed (see Figure 3-25).

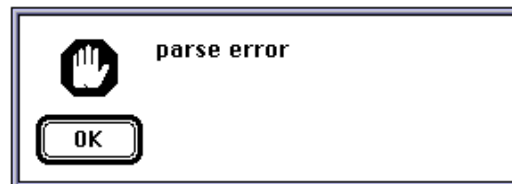


Figure 3-25: Error Box Showing a Parse Error (Macintosh)

9. The error was intentional to show you how to edit and debug your programs. Click on the OK button or press RETURN.
10. Pull down the “**Windows**” menu and select “**Locate Tutorial**” at the bottom of the menu.

NOTE: Always remember that the error will be in the previous line selected. The second line is selected in our tutorial program because the first line has the error in it.

11. Using the mouse click in between the “x” and the word “for”. Press the SPACE bar to enter a space between the “x” and the word “for”.
12. Enter Command-Y (which is the Syntax Check command). This time Figure 3-26 should be displayed.

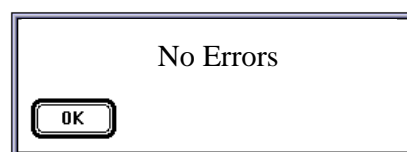


Figure 3-26: No Errors Box (Macintosh)

The “No Errors” box verifies that the program has no syntax errors. (In technical terms, the Syntax Check command *tokenizes* the program in the edit buffer and reports errors but does not attempt to load the program into the Tattletale.) Press the RETURN key or click on the OK button.

NOTE: Just because the “No Errors” box is displayed, it does not mean there are no errors in your program, it only means there are no errors in the syntax of the commands used in your program.

13. Enter Command-R (which means RUN) to instruct the Tattletale to run the program currently in RAM (which right now is our 3 line program). The Tattletale will tokenize the program and load it into the Tattletale. It will then switch to the Terminal window and display “Hello” ten times.

Congratulations! - You have written, debugged and run your first TxBASIC program!

This completes the TxTools tutorial. The other TxTools commands are explained in detail in the “[Explanations of TxTool Menu and Window Options \(Macintosh\)](#)” on page 3-25. Please proceed to [Section 4 - Using TxBASIC](#) for a tutorial on using TxBASIC and refer to [Section 5 - TxBASIC Command Reference](#) for detailed explanations of each TxBASIC command.

Keyboard Shortcuts for Mouse Actions (Macintosh)

Table 3-12 shows all the keyboard commands that are usually operated by using the mouse. Macintosh computers always have a mouse; however, many users find using the keyboard faster than always reaching for the mouse to enter a command.

Table 3-12: Keyboard Shortcuts for Mouse Actions (Macintosh)

Menu Selections			
Menu Option	Command Key	Menu Option	Command Key
New	Command-N	Find	Command-F
Open	Command-O	Enter Selection	Command-E
Save	Command-S	Find Again	Command-G
Get Info	Command-I	Replace	Command-=
Quit	Command-Q	Replace and Find Again	Command-H
Undo	Command-Z	Compare Windows	Command-M
Cut	Command-X	Run	Command-R
Copy	Command-C	Load	Command-L

Table 3-12: Keyboard Shortcuts for Mouse Actions (Macintosh) (Continued)

Menu Option	Command Key	Menu Option	Command Key
Paste	Command-V	Syntax Check	Command-Y
Paste Date/time	Command-D	Hide Terminal Window	Command-T
Shift Left	Command-[Clear Buffer	Command-B
Shift Right	Command-]	Send Break	Command-/
Invert Case	Command-U		
Dialog Box			
Action	Key to Press	Action	Key to Press
Cancel	Escape or Command-Period	OK	Return
Editing Controls			
Action	Key to Press	Action	Key to Press
Move cursor	Arrow keys	Move cursor a page	PgUp, PgDn
Move to line start	Home	Move to line end	End
Delete character	Backspace or Delete		

Explanations of TxTool Menu and Window Options (Macintosh)

Introduction (Macintosh)

This part of the manual describes the TxTools windows and the various areas of the windows you need to understand to work with TxTools. A detailed description of each menu and sub-menu options are also included.

Explanation of TxTool Window Types (Macintosh)

Terminal Window (Macintosh)

When you first start TxTools, you will immediately see one large window (see Figure 3-27 and Table 3-13). This window displays any characters that are received by the serial port (including any characters you've typed that the Tattletale has echoed back).

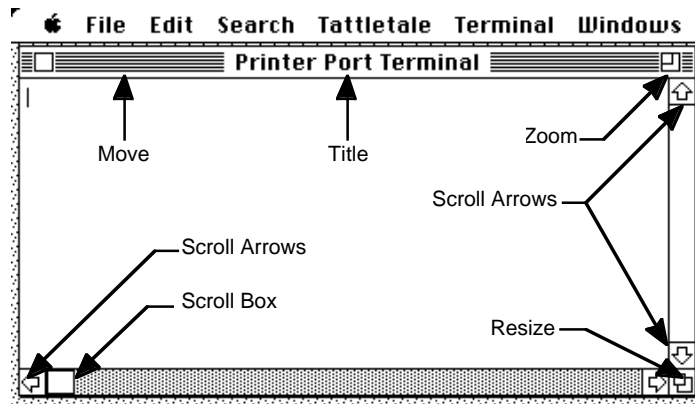


Figure 3-27: Terminal Window Display (Macintosh)

Table 3-13: Terminal Window Feature Descriptions (Macintosh)

Feature	Description
Move	The Move arrow points to the top frame of the window. You can move the window by clicking and holding anywhere along this top frame and then moving the mouse. The window frame will follow the mouse until you release the button.
Title	The Terminal window is always either "Modem Port Terminal" or "Printer Port Terminal". The Edit file window will be named whatever the file was saved as (or "Untitled" if it has not been saved yet).
Zoom	By clicking on the Zoom box, you can toggle the size of the window between full screen and a smaller size.
Scroll Arrows	Scroll arrows cause the text in the window to scroll one line in the direction of the arrow. Click and hold on these arrows to scroll continuously.
Scroll Box	You can move more quickly through a document by click-hold-dragging on the box. Moving the box toward the top moves closer to the start of the document and moving the box toward the bottom moves closer to the end of the document. The window doesn't scroll until the box is released.
Resize	The size of the window can be adjusted by click-hold-dragging on the Resize corner of the window frame.
<p>NOTE: The Terminal window has a finite sized circular buffer. The default 8000 byte buffer holds about 8 pages of packed text. When the buffer fills, the oldest characters are scrolled off the top and lost. Refer to Figure 3-41 on page 3-41 for further information on changing the review buffer size. Also refer to Table 3-22 on page 3-41 for further information.</p>	

Edit File Window Description (Macintosh)

Figure 3-28 is an Edit File window. Title is the name of the program file. Notice the Close box in the upper left corner. The Close box attempts to close the active front window. You will be asked if you want to save the changes if the window text has been modified since the last save. You can agree to save changes, throw out all changes since the last save or cancel the close.

Characters are entered at the flashing cursor. Backspace or delete removes characters before the cursor.

To select a block of text, click and hold the mouse button over the beginning of the block and drag the mouse to the end of the block. When you release the mouse button, the block will be selected. You can then cut, copy, paste or clear this block. Moving the cursor causes the selection to be lost, however. The editor also has an Undo capability to reverse the last action taken. Windows continue to exist and be displayed even if they are not the front active window.

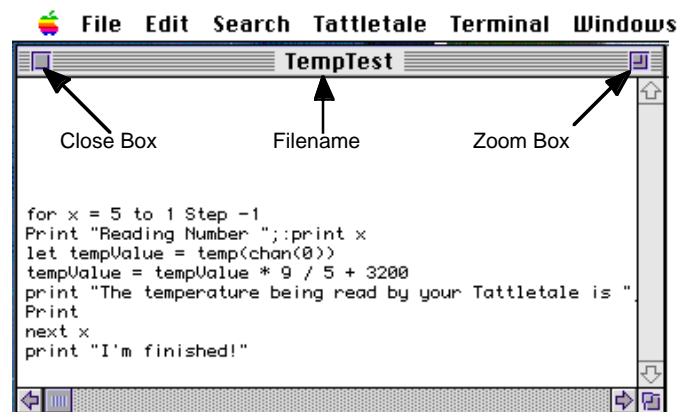


Figure 3-28: Edit File Window (Macintosh)

Apple Menu Option Descriptions (Macintosh)

This item contains 2 sub-selections (see Figure 3-29 and Table 3-14).

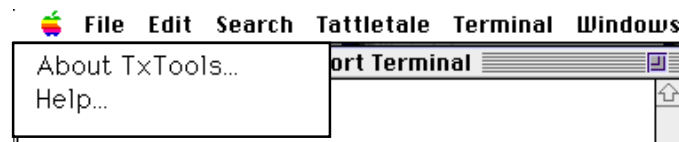


Figure 3-29: Apple Menu Options (Macintosh)

Table 3-14: Apple Menu Option Descriptions (Macintosh)

Apple Menu Options	Description
About TxTools	Selecting this item displays a dialog showing the version number of TxTools you are using.
Help	A list of TxTools menus and the command key combinations that trigger them. When you're done with the help system, either click on the Close box or press the ESCAPE key.

File Menu Option Descriptions (Macintosh)

The File menu contains 12 sub-selections (see Figure 3-30 and Table 3-15).

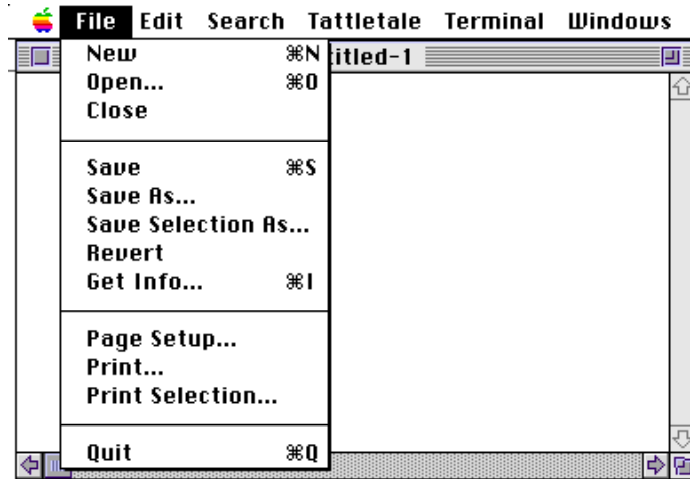


Figure 3-30: File Menu Options (Macintosh)

Table 3-15: File Menu Option Descriptions (Macintosh)

File Menu Option	Description
New (Command-N)	Opens a new “Untitled” window to enter your program into.
Open (Command-O)	<p>When you select this option the dialog box shown in Figure 3-31 is displayed. You simply select the TxTools program file you want to open and click on the OPEN button or double click on the file name. See Figure 3-28 for an example of a TxTools program file. The text of the file is viewed in the Edit File window (see Figure 3-28).</p> <p>To Cancel out of this dialog, you have two choices: click on the Cancel button (under the Open button) or press the Escape key.</p>
Close	Closes the active front window. If changes have been made in the file since it was opened, you will be asked if you want to save the changes before closing.
Save (Command-S)	<p>Saves the program file in the active window to disk. If this is a new file, you will be prompted for a file name to save the program as.</p> <p>NOTE: The file will be automatically saved everytime the “Run” command is used from the Edit file window. If the file has never been saved before, the “Save As” dialog box will be displayed.</p>

Table 3-15: File Menu Option Descriptions (Macintosh) (Continued)

File Menu Option	Description
Save As	Allows you to save the program in the active window to disk under a new name (see Figure 3-24). This is useful when you wish to experiment with your program, but you don't want to make the changes permanent to your original file.
Save Selection As	Allows you to select some text anywhere on the screen and save it as a TxTools program file.
Revert	Forces the program to open the last saved version of your program file while deleting the version currently in memory. TxTools will ask if you want to revert to the last version saved before you delete the current file in memory.
Get Info (Command-I)	Displays the numbers of lines in the program, the size of the program file in bytes and the amount of memory still available. There is a read-only check box to protect the file from anyone making changes to it by mistake.
Page Setup	Displays the standard Macintosh Page Setup dialog box for printing.
Print (Command-P)	Sends the text of the current active window to the printer that is selected in the chooser. The standard Macintosh print dialog box is displayed before starting to print.
Print Selection	Sends any currently selected text to the printer. The standard print dialog box is displayed before printing is started. If no text is selected, this option will be dimmed so that it can't be selected.
Quit (Command-Q)	Exits TxTools and returns to the Finder.

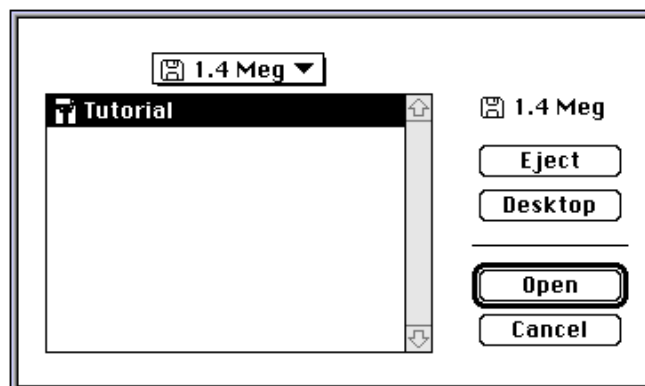


Figure 3-31: Open File Dialog Box (Macintosh)

Edit Menu Option Descriptions (Macintosh)

The Edit menu contains 11 sub-selections (see Figure 3-32 and Table 3-16).

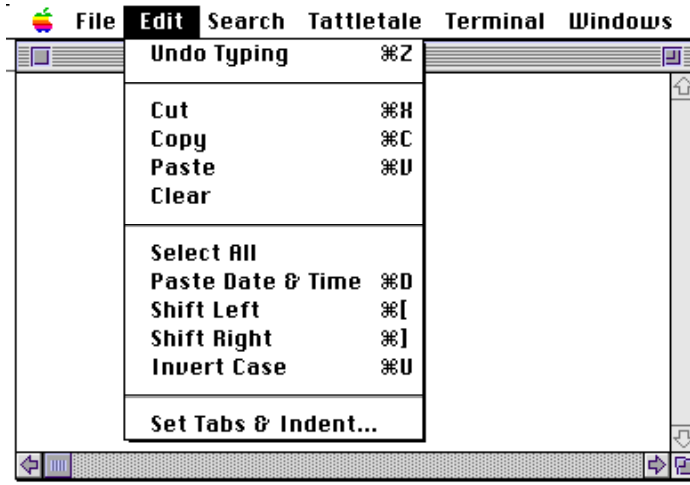


Figure 3-32: Edit Menu Options (Macintosh)

Table 3-16: Edit Menu Option Descriptions (Macintosh)

Edit Menu Option	Description
Undo (Command-Z)	Select this item to undo an editing action. Only the last action performed can be undone.
Cut (Command-X)	Remove the currently selected text and save it on the clipboard for pasting later.
Copy (Command-C)	Save a copy of the currently selected text on the clipboard for pasting later. Unlike Cut, this does not remove the selected text.
Paste (Command-V)	Insert whatever is on the clipboard at the cursor. Use Cut or Copy to get text onto the clipboard.
Clear (Delete key)	Remove the currently selected text without saving a copy on the clipboard. You can Undo a Clear but not after another action is performed.
Select All	Selects all the text in the front active window.

Table 3-16: Edit Menu Option Descriptions (Macintosh) (Continued)

Edit Menu Option	Description
Paste Date & Time (Command-D)	Insert a text string showing the current date and time of your computer's clock into the document. If the Terminal window is currently the active window, then this option will not be available. Selecting this item sends a string of the form: Fri, Sep 30, 1994, 3:55 PM. If you hold the Shift key down while selecting this, a longer date/time string of this form is used: Friday, September 30, 1994, 3:56 PM.
Shift Left (Command-L)	This command will move the entire line of selected text one space to the left (no farther than the left margin). Some text of at least one line must be selected for this command to be available.
Shift Right (Command-R)	This command will move the entire line of selected text one space to the right. Some text of at least one line must be selected for this command to be available.
Invert Case (Command-U)	Any text that is selected will have the case switched by this command: All upper case letters will become lower case and all lower case characters will become upper case.
Set Tabs & Indent	Allows you to set the number of spaces the cursor will move to the right when the tab key is pressed.

Search Menu Option Descriptions (Macintosh)

The Search menu contains 9 sub-selections (see Figure 3-33 and Table 3-17).

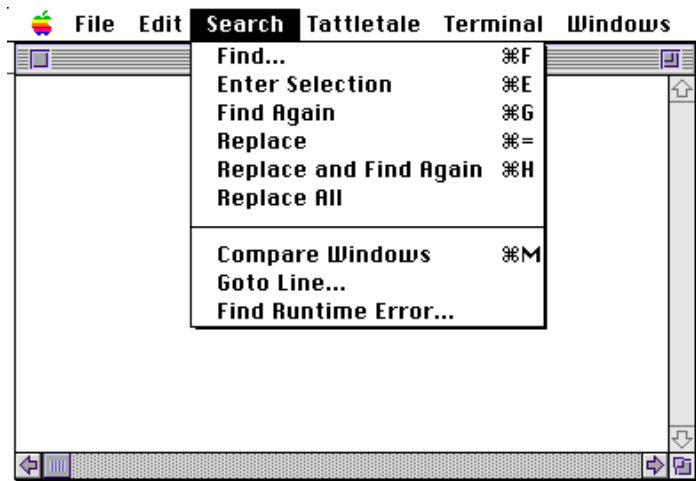


Figure 3-33: Search Menu Options (Macintosh)

Table 3-17: Search Menu Option Descriptions (Macintosh)

Search Menu Option	Description
Find (Command-F)	A dialog box will appear to allow you to enter the text to look for in the document (Figure 3-34). Type the text to search for into the box labeled "Find". To the right of the Find box is a box labeled "Change" (the replace command uses this dialog box also). The change box is where you enter the new text that will be used to replace the Find text. See Table 3-18 for a description of all the options in the Find dialog box.
Enter Selection (Command-E)	If you have text selected and you choose this option the text will automatically be copied into the Find dialog box. You can then use the Find Again command without actually going into the Find dialog box.
Find Again (Command-G)	Once a text string has been found with the Find command, you can continue to look through the document for other occurrences of the same text string. It is usually easier to use the Command-G keyboard equivalent for this command.
Replace (Command-=)	If text has been entered into the Find and Change To dialog box, this command will replace the text found with the text from the Change To box. The cursor will not change position after changing the text. NOTE: Entering nothing in the "Change To" box means that found text will be erased.
Replace and Find Again (Command-H)	Same as the Replace command; however, the next occurrence of the text to find will automatically be selected.
Replace All	Finds every occurrence of the "Find" text and automatically replaces it with the new text from the "Change To" box. NOTE: This command does not have an Undo function.
Compare Windows (Command-M)	If you open two versions of your program file and select this option, the differences between the two program files will be displayed. No changes will be made to either version of your programs.
Goto Line	Moves the cursor to whatever line number you enter. NOTE: Every line counts as a number, even if there is no text on the line.

Table 3-17: Search Menu Option Descriptions (Macintosh) (Continued)

Search Menu Option	Description
Find Runtime Error...	When you have a “How” error it displays two numbers that are the “Error I.D.” and the “Token Number”. This command opens a dialog box requesting the “Error I.D.” and the “Token Number”. After entering the necessary data in these two boxes the location of your error will be shown. This command is for debugging purposes only.

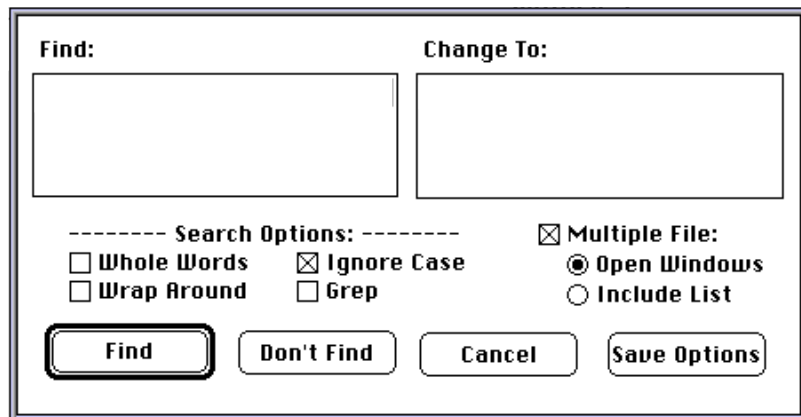


Figure 3-34: Find Option Dialog Box (Macintosh)

Table 3-18: Find Dialog Box Option Descriptions (Macintosh)

Find Dialog Box Options	Description
Whole Words	Will only find whole words in the document. If you searched for the word “cat” with this option checked, the word “catalog” would not be found. If this option was not checked, then the word “catalog” would be found.
Wrap Around	With this option checked, you can start searching from anywhere in the document and all occurrences of the word will be found anywhere in the document. With this option not checked, only the text between the cursor and the end of document will be searched.
Ignore Case	If this option is checked, then any occurrence of the text to find will be found regardless of whether the case of the text to find is different than the text found in the document.
Grep	(Global Regular Expression Print) The user specifies a pattern to search for. This utility searches the input file line by line, checking each one to see if it contains the search text.

Table 3-18: Find Dialog Box Option Descriptions (Macintosh) (Continued)

Find Dialog Box Options	Description
Multiple File	If this option is checked and the Open Windows sub-option is selected, then the text in the “Find” field will be searched for in all open windows (except the Terminal window). If the “Include List” sub-option is selected, then the user will be prompted for which files to search through.
Save options	Permanently saves all the options selected for the next Find action.
Don't Find	This closes the Find dialog box without losing the search text in the Find dialog box.

Tattletale Menu Option Descriptions (Macintosh)

The Tattletale menu option contains 11 sub-selections (see Figure 3-35 and Table 3-19).

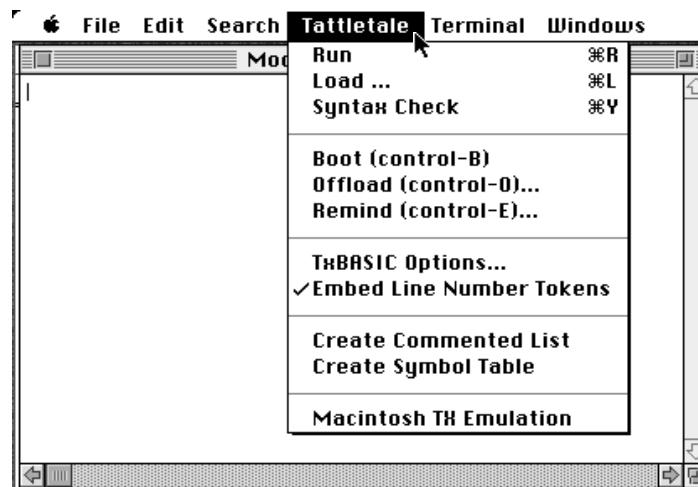


Figure 3-35: Tattletale Menu Options (Macintosh)

Table 3-19: Tattletale Menu Option Descriptions (Macintosh)

Tattletale Menu Option	Description
Run (Command-R)	<p>If selected from the Terminal window, a RUN command is sent to the Tattletale to execute the program currently in RAM. If selected from an edit window, the program is first checked to see if it has been changed. If it has been changed, it is tokenized and loaded into the Tattletale. Then the Terminal window is brought to the front as the active window and a RUN command is sent to the Tattletale.</p> <p>NOTE: The file will be automatically saved everytime the “Run” command is used from the Edit file window. If the file has never been saved before, the “Save As” dialog box will be displayed.</p>
Load (Command-L)	<p>Causes the program in the edit buffer to be tokenized and loaded into the Tattletale. The Terminal window is brought to the front as the active window.</p>
Syntax check (Command-Y)	<p>Tokenizes the program in the edit buffer and reports errors but does not attempt to load the program into the Tattletale. This is affected by the “TxBASIC Options” menu selections (see Figure 3-36 and Table 3-20).</p>
Boot EPROM	<p>Loads the program burned in the EPROM into RAM and executes it. This also sets up the Tattletale to execute the EPROM program automatically the next time the Tattletale is reset. When you choose this item when an edit window is selected, TxTools automatically switches to the Terminal window before booting the EPROM. This works exactly the same for the Model 5F, Model 5F-LCD and the Model 6F which use an EEPROM.</p>
Offload Datafile	<p>Presents you with a dialog box (see Figure 3-37) that allows you to choose how to off-load the datafile of the Tattletale. To off-load part of the datafile, be sure to fill out the "Start address" and "End address" boxes and click the "Partial" button (or press the RETURN key on the keyboard). For the entire datafile just click the "Off-load All" button. After finishing this dialog box, a second dialog box allows selecting the name of the file. You can save it anywhere on the disk.</p>

Table 3-19: Tattletale Menu Option Descriptions (Macintosh) (Continued)

Tattletale Menu Option	Description
Remind EPROM	This command no longer works for the Macintosh version of TxTools. If you need to get a copy of your program and TxBASIC in hex format, you will need to use TxTools on an IBM PC or compatible. The resulting hex file can then be transferred to the Tattletale using either the Macintosh or IBM PC version of TxTools.
TxBASIC Options	Dialog box with flags that can be set to change the tokenizer, the loading process, the off-loading process and the Remind EPROM process (see Figure 3-36 and Table 3-20).
Embed Line Number Tokens	Causes the tokenizer to insert a line number in the object program for each line in the program file. This can help in later debugging but it adds three bytes per line to the size of the program. NOTE: The line numbers are not displayed.
Create Commented List	The next time the tokenizer runs (whether through the Load command or the Syntax check command), an annotated listing of the program is stored in a text file. The file saved will have the extension LST added to the filename. The text file created is useful for debugging programs.
Create Symbol Table	The next time the Syntax check or Load command is used, a symbol table will be saved on your disk with the extension SYM added to your filename. A symbol table is always created when you choose the Load command. This can help with debugging a program. It can also help with checking for consistent spelling of all your variables.
Macintosh TX Emulation	This item must be unchecked for TxTools to operate properly. For information on making this state permanent, see the “Macintosh TXBASIC Emulation” entry in Table 3-20.

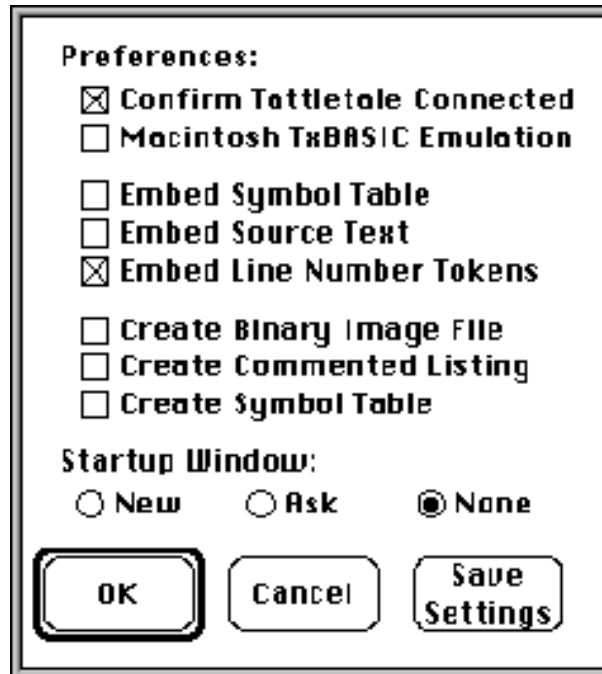


Figure 3-36: TxBASIC Options Sub-Menu (Macintosh)

Table 3-20: TxBASIC Options Sub-Menu Descriptions (Macintosh)

TxBASIC Options Dialog Box	Description
Confirm Tattletale Connected	Forces TxTools to wait for a reply from the Tattletale before starting a program load, a datafile off-load or a Remind EPROM.
Macintosh TxBASIC Emulation	This item must be unchecked for TxTools to operate properly. To make your preferences permanent, click the “Save Settings” button after you have set the preferences to the state you want.
Embed Symbol Table	Function not available.
Embed Source Text	Function not available.
Embed Line Number Tokens	Causes the tokenizer to insert a line number in the object program for each line in the source program. This can help in later debugging but adds three bytes per line to the size of the program. NOTE: The line numbers are not displayed.

Table 3-20: TxBASIC Options Sub-Menu Descriptions (Macintosh) (Continued)

TxBASIC Options Dialog Box	Description
Create Binary Image File	The next time the tokenizer runs (whether through the Load command or the Syntax check command), an exact copy of the program (as it's loaded into the Tattletale) is stored in a binary file. The file saved will have the extension BIN added to the filename.
Create Commented Listing	The next time the tokenizer runs (whether through the Load command or the Syntax check command), an annotated listing of the program is stored in a text file. The file saved will have the extension LST added to the filename. The text file created is useful for debugging programs.
Create Symbol Table	The next time the Syntax check or Load command is used, a symbol table will be saved on your disk with the extension SYM added to your filename. A symbol table is always created when you choose the Load command. This can help with debugging a program. It is also helpful when checking for consistent spelling of all your variables.
Startup Window New Ask None	"New" creates a new "Untitled" window everytime TxTools is started. "Ask" causes the program to startup with the open file dialog box displayed. "None" forces the program to go directly to the Terminal window.

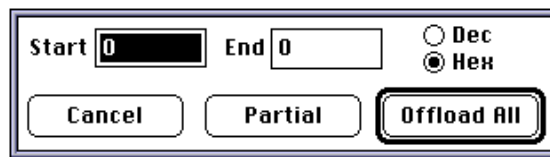


Figure 3-37: Off-load Datafile Option Dialog Box (Macintosh)

Terminal Menu Option Descriptions (Macintosh)

The Terminal menu option contains 9 sub-selections (see Figure 3-38 and Table 3-21).

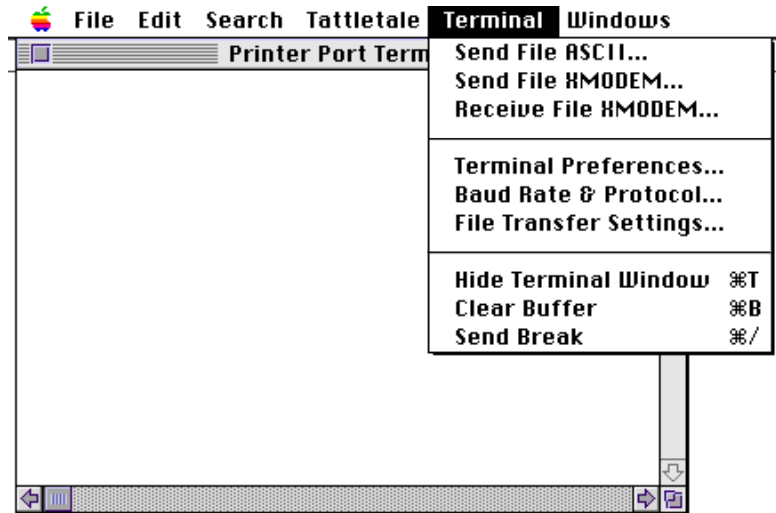


Figure 3-38: Terminal Menu Options (Macintosh)

Table 3-21: Terminal Menu Option Descriptions (Macintosh)

Terminal Menu Options	Description
Send File ASCII	Allows you to send a text file out of the serial port to the Tattletale. See Figure 3-39 for the Send File ASCII dialog box.
Send File XMODEM	Allows you to send a text or binary file out the serial port to the Tattletale. See Figure 3-39 for the Send File XMODEM dialog box.
Receive File XMODEM	Starts the XMODEM receive process for a text or binary file coming into the serial port. See Figure 3-40 for the Receive File XMODEM dialog box. NOTE: You do not need this when off-loading the Tattletale. The "Off-load datafile" command and the "Remind EPROM" command do this for you automatically.
Terminal Preferences	Refer to Figure 3-41 and Table 3-22 for a description of each terminal preference option.
Baud Rate & Protocol	Displays all the communication settings (see Figure 3-42)

Table 3-21: Terminal Menu Option Descriptions (Macintosh) (Continued)

Terminal Menu Options	Description
File Transfer Settings	<p>These only affect ASCII file transfers and program loads. If Char delay is not zero, it adds the specified number of milliseconds between characters to both ASCII file transfers and program loads (see Figure 3-43). If Line Delay is not zero, it adds the specified number of milliseconds after sending a carriage return for ASCII file transfers only. The “Send Prefix” option sends the prefix text before sending a file to the Tattletale.</p> <p>The “Send Suffix” option sends the suffix text after sending a file to the Tattletale.</p> <p>The “Discard Echoed Text” option prevents all echoed text from the Tattletale from being displayed in the Terminal window.</p>
Hide Terminal Window (Command-T)	<p>Hides the Terminal window. Since this window is not an edit window, you will not be prompted to save. The “Locate Terminal Window” command, under the Windows menu, will re-open the Terminal window.</p>
Clear Buffer (Command-B)	<p>Erases all text currently displayed in the Terminal window.</p>
Send Break (Command-/))	<p>This command applies only to the Model 6F. If the 6F is placed in a very low power state with the HYB or QUIT command, the Send Break menu option will force the 6F to exit these commands and continue with the next step in the program.</p> <p>NOTE: Do not confuse this command with Cntl-C which stops executing the program completely.</p>

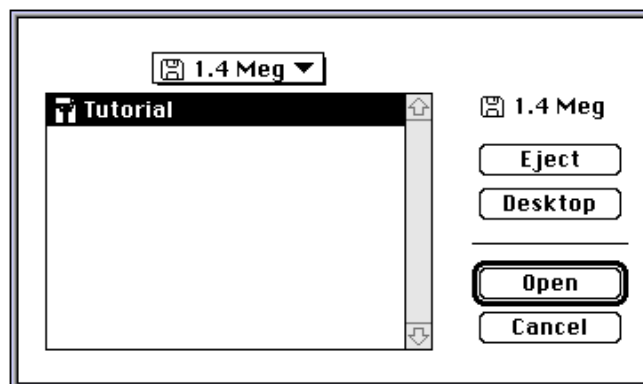


Figure 3-39: Send File ASCII or Send File XMODEM Dialog Box (Macintosh)

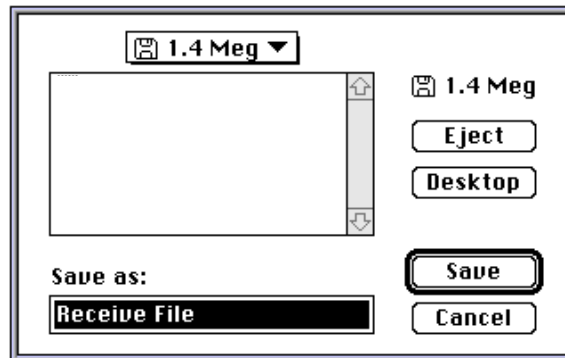


Figure 3-40: Receive File XMODEM Dialog Box (Macintosh)

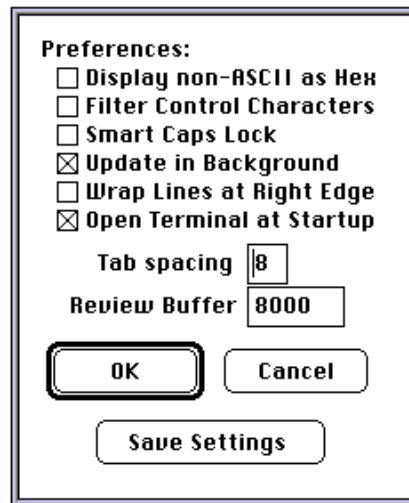


Figure 3-41: Terminal Preferences Sub-Menu (Macintosh)

Table 3-22: Terminal Preferences Sub-Menu Descriptions (Macintosh)

Preference Name	Description
Display non-ASCII as Hex	Displays all non-standard ASCII characters (such as control characters) as hex numbers instead of displaying garbage characters.
Filter Control Characters	Discards unprintable control characters.
Smart Caps Lock	With this option selected, TxTools always will startup with all characters being typed as capitals. After starting up, pressing the Caps Lock key will toggle the case as normal.

Table 3-22: Terminal Preferences Sub-Menu Descriptions (Macintosh) (Continued)

Preference Name	Description
Update in Background	With this option selected, the Terminal window will continue to receive and display information even if the Terminal window is not the front active window.
Wrap Lines at Right Edge	Causes any long text strings that are displayed while running a program to be wrapped back to the left edge automatically. The wrapping is directly affected by the size of the Terminal window. NOTE: This option has no effect on your program in the Edit file window.
Open Terminal at Startup	With this option selected, everytime TxTools is started the Terminal window will open automatically.
Tab spacing	Allows the user to select how many spaces each Tab will move to the right.
Review Buffer	Specifies the size of the Terminal window review buffer, beyond which scrolled characters are lost off the top. Increasing this number may require increasing the minimum memory requirements for TxTools from the finder.

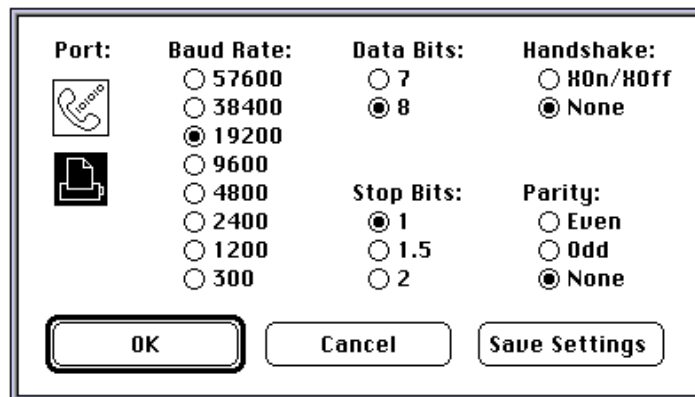


Figure 3-42: Baud Rate / Protocol Option Dialog Box (Macintosh)

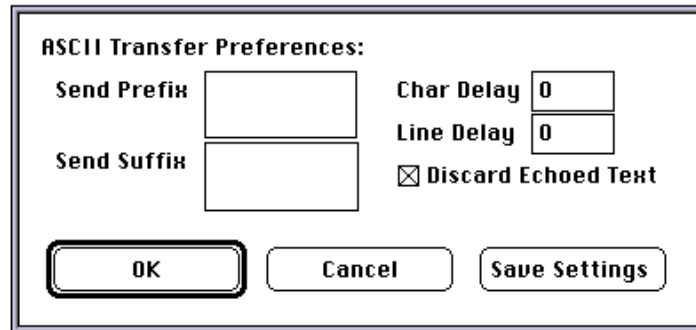


Figure 3-43: ASCII Transfer Option Dialog Box (Macintosh)

Windows Menu Option Descriptions (Macintosh)

This item contains 8 standard sub-selections (see Figure 3-44 and Table 3-23) and the names of any open program files.

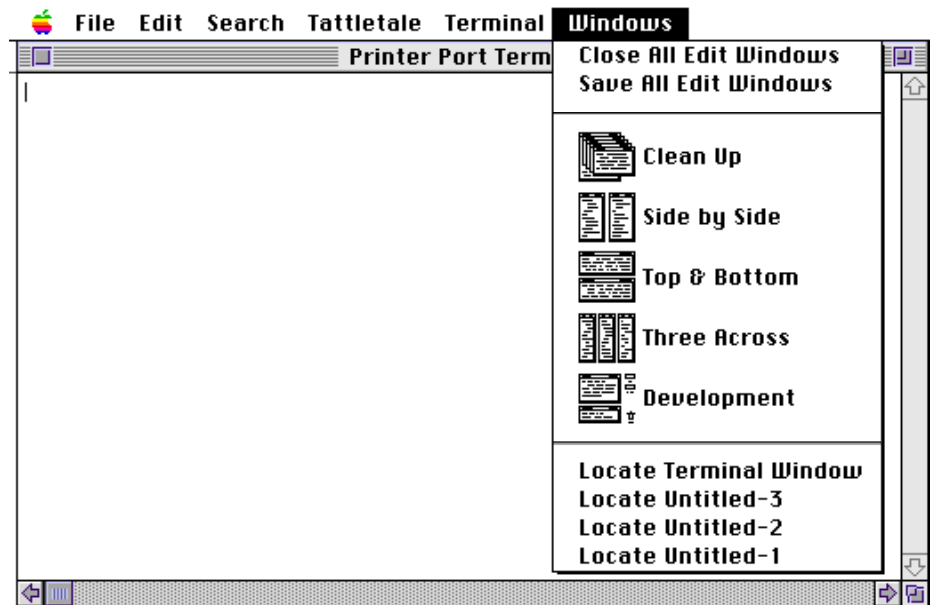


Figure 3-44: Windows Menu Options (Macintosh)

Table 3-23: Windows Menu Option Descriptions (Macintosh)

Windows Menu Options	Description
Close All Edit Windows	Closes all windows except the Terminal window. If you made changes since the last time you saved, the program will prompt you with the option to save before the file is closed.
Save All Edit Windows	Saves changes in all the edit file windows.
Clean Up	Aligns all open windows so that the title bar of each is visible.

Table 3-23: Windows Menu Option Descriptions (Macintosh) (Continued)

Windows Menu Options	Description
Side by Side	Puts the last two open windows side -by- side, filling the entire screen.
Top & Bottom	Puts the last two open windows in the top and bottom half of the screen, filling the entire screen.
Three Across	Arranges the last three open windows into three equal columns, filling the entire screen.
Development	Puts the Terminal window at the bottom of the screen and one program file in the top half of the screen.
Locate Terminal Window	Brings the Terminal window to the front as the active window.

Software Change Information for TxTools (IBM PC and Macintosh)

All recent software changes are documented in the Read Me file that came on your distribution disk.

Program Parameters Saved in the Configuration File (IBM PC Only)

The following program parameters are saved in the TXTOOLS.CFG file and will be recalled the next time the program is started:

- Comm Port number (1, 2, 3 or 4)
- Comm Port baud rate (300, 1200, 2400, 4800, 9600, 19200, 38400 or 57600)
- Comm Port parity setting (none, even or odd)
- Comm Port stop bits (1 or 2)
- Comm Port data bits (7 or 8)
- Comm Port flow control setting (XON/XOFF control can be enabled or disabled)
- Screen Mode can be low resolution or high resolution (for EGA and VGA)
- ASCII file transfer, amount of delay between characters (not normally needed)
- ASCII file transfer, amount of delay at end of line (not normally needed)
- Whether capture mode overwrites an existing file or appends to it
- The five user-defined file extensions used in the 'open file' dialog
- The settings of the Tattletale | Options tokenizer flags (on or off)
- The size of the Tattletale | Options "Maximum Symbols" value
- The size of the Tattletale | Options "Maximum ASM Lines" value
- The CommPort | Options "Send Prefix" string
- The CommPort | Options "Send Suffix" string

Section 4 - Using TxBASIC

Introduction

TxBASIC is the operating system used to control the functions of the Tattletale. All of the commands and procedures shown in this section are entered through the TxTools program which acts as the development area. For an explanation of how TxTools interacts with TxBASIC, refer to [Section 3 - Operating the TxTools Program](#). This section and [Section 5 - TxBASIC Command Reference](#) are to be used primarily for reference. A tutorial showing the basics is included and many of the commands have example programs for clarity.

General TxBASIC Information

Abbreviations:

There are no abbreviations for any TxBASIC commands.

Arrays:

The number of arrays in TxBASIC is limited only by the size of the variable storage area. The ADLOOP array and “?” array are predefined. Refer to the DIM command on [page 5-31](#) for additional information. We now allow two dimensional arrays. They are declared just like one-dimensional arrays with an extra value enclosed in parentheses. Example: `dim array(5, 7)`.

Arrays of string variables are not allowed.

Arithmetic Operators:

The five arithmetic operators have the highest priority of all of the TxBASIC operators.

NOTE: TxBASIC does not have a separate high priority unary minus operator, but instead treats the negation of a constant or variable as zero minus the value of the variable or constant.

The TxBASIC arithmetic operators are listed in Table 4-1 in the order of precedence (all operators on the same level are evaluated left to right). Refer to [“TxBASIC Floating Point”](#) on page 4-46 for details about arithmetic operations involving floats.

Table 4-1: TxBASIC Arithmetic Operators

Precedence Level	Arithmetic Operator	Used For
Highest Precedence	* / %	Multiplication, Division, Modulo
	+ -	Addition, Subtraction
	> <= > >= <> =	Relational operators
	&	Logical bitwise AND
Lowest Precedence		Logical bitwise OR

Assembly Language:

The TxBASIC tokenizer has a built-in full featured 6303 assembler. Code can be assembled in line with the program or into a separate area as desired.

NOTE: Assembly language is not available on the Model 8.

Break:

CTRL-C can break a program in TxBASIC. A special command 'CBREAK' followed by a label can be used to specify the address to restart to when a CTRL-C is received. CTRL-C will only break a program running in the foreground. Refer to “**Dual Tasking**” on [page 4-25](#) and the “**Run**” command on [page 5-84](#). You can disable CTRL-C breaks by writing a zero byte to address 9E hex. A count of CTRL-C characters will continue to be updated at address 93 hex. Clear this before re-enabling break-outs. Refer to the CBREAK command on [page 5-21](#) for additional information.

Case:

Labels and variable names are case sensitive, commands and keywords are not case sensitive.

Comments:

TxBASIC provides three ways to include comments in your code. 'REM' causes the rest of the command to be ignored. An apostrophe (') at the beginning of a line causes the entire line to be ignored. A pair of slashes (//) can appear anywhere on the line and cause the rest of the line to be ignored. Assembly code sections use “;” for comments.

Constants:

TxBASIC supports string, integer, floating point and character constants (1-4 characters in a 4-byte integer). Integer constants are signed decimal numbers in the range -2147483647 to +2147483647 or unsigned hexadecimal numbers in the range &H0 to &HFFFFFFF. Floating point constants must include a decimal point and/or the power-of-ten specifier 'E'. All floating point constants are single precision with a range of $\pm 1.175494E-38$ to $\pm 3.402823E+38$ and 0.0. Refer to “**TxBASIC Floating Point**” on [page 4-46](#) for details. There are no short integer constants, and no octal constants.

String constants must be bracketed by double quote characters. Strings can contain 0-255 characters. String constants can be used anywhere a string variable is used except on the left side of an assignment operator.

We now allow character constants where up to four characters are enclosed in single quotes. For instance, 'A' is a character constant equivalent to value 41 H. 'AB' is equivalent to 4142 H. 'ABC' is equivalent to 414243 H. 'ABCD' is equivalent to 41424344 H.

Datafile: Storage and Retrieval

The Tattletales have a data storage area called the “datafile”, which can be reached by several commands, using any variable as a pointer. The datafile is not altered except by a data storage command or when power is removed from the logger.

Datafile commands are ADLOOP, BURST, DFREAD, DFSAVE, GET, GETS, ITEXT, OTEXT, STORE, UGET, USEND and OFFLD, and in special forms of PRINT, CALL, SDO and STORE. All commands that access the datafile automatically increment the pointer variable to point to the next location.

Division by Zero:

Integer division by zero will cause the program to stop executing and display the "HOW" error message. Refer to [“Tattletale Error Messages”](#) on page 4-24 for more information on error handling. Floating point division by zero does not stop program execution. As will other floating point errors, it sets a bit in the FPERR error variable to indicate an error. Division by zero returns a result of infinity. Refer to [“TxBASIC Floating Point”](#) on page 4-46.

Editing:

Editing is done on the host computer using TxTools. The Tattletale itself has no facilities for editing.

Errors:

A list of error statements and their causes can be found in [“Tattletale Error Messages”](#) on page 4-24. An error will stop program execution and display a message unless an ONERR command has been executed by the program. Refer to the ONERR command on [page 5-65](#) for additional information.

Floating point:

Single precision floating point math is available in TxBASIC along with a number of trigonometric functions. Refer to [“TxBASIC Floating Point”](#) on page 4-46 for more information.

Hexadecimals:

Hexadecimal numbers can be entered by preceding the number with '&H'. The entered number must be unsigned and may include up to eight hexadecimal characters. It will be treated as a signed 32-bit two's-complement number internally. In assembly code, hexadecimals may also be entered preceded by an “H”.

Analog / Digital I/O:

Standard dialects of BASIC use the keyboard and disk storage for data input. In the Tattletale, these inputs are augmented by the logger's analog and digital inputs. A number of new commands and functions have been designed to deal with these inputs and outputs efficiently. The analog commands are ADLOOP, CHAN and BURST. The digital I/O commands are PIN, PSET, PCLR, PTOG, COUNT, PERIOD, SDI and SDO.

Labels, Assembler:

Labels can be used in the assembly code for flow control and to define local variables. Assembler labels can be up to 32 characters long, must begin with a letter or an underscore (_), and end with a colon. The only valid characters in a label are upper and lower case characters, the numbers and underscore. The label name must start in the first column of the line. Assembler label names are not required to be terminated with a colon when defined.

Labels, TxBASIC:

TxBASIC labels can be up to 32 characters long, must begin with a letter, an underscore (_) or the @ symbol and end with a colon. The only valid characters in a label are upper and lower case characters, the numbers, underscore and @. TxBASIC labels do NOT have to begin in the first column of the line.

Line Forms:

There are almost no immediate commands and line numbers can NOT be used. Blank lines are permissible.

Line Numbers:

Line numbers can NOT be used in TxBASIC. You can however have a one character label followed by a number to simulate a line number. For example: M1: could be used as the line number for line one of your program (TxBASIC would use it as a label).

Logical Operators:

The Tattletale supports the two logical operators, AND and OR, which are used for both bit-wise operations and logical connectives. Unlike most other BASIC dialects, the operators are not spelled out, but instead are represented by the symbols "&" for AND, and "|" for OR.

Multiple Statements:

TxBASIC supports the use of colons to allow multiple statements on a single program line. GOSUBs on a multiple statement line return to the statement following the GOSUB on the same line. REM statements on a multiple statement line cause the remainder of the line to be ignored (even if the line contains more statements separated by colons). Obviously, a GOTO command in a multiple statement line will prevent whatever follows the GOTO from ever being executed. A space in front of a colon, used as a separator, will prevent it from being mistaken as a label.

Overflow:

Overflow errors are detected during the evaluation of an expression when the intermediate value becomes greater than the maximum long integer (four byte integer) value of 2,147,483,647, or less than the minimum long integer value of -2,147,483,648. Notice that TxBASIC variables can actually hold one more negative value than can be expressed as an integer constant. Overflow or underflow can occur in floating point numbers if the intermediate value is outside the range $\pm 1.175494\text{E}-38$ to $\pm 3.402823\text{E}+38$. Refer to [“TxBASIC Floating Point”](#) on page 4-46 for details.

Integer overflow errors cause the program to stop executing and display the "HOW" error message. Refer to [“Tattletale Error Messages”](#) on page 4-24 for more information on error handling.

Relational Operators:

TxBASIC supports seven relational operators which are used for comparing two values. Relational operations return 1 if the result of the comparison is true, and 0 if the result is false. Refer to [“TxBASIC Floating Point”](#) on page 4-46 for details about comparisons involving floats. Strings can be compared with other strings (both constants and variables) but cannot be compared with numerical values. The relational operators are shown in Table 4-2.

Table 4-2: Relational Operators

Operator	Meaning	Usage
<	less than	A<B
<=	less than or equal to	A<=B
>	greater than	A>B
>=	greater than or equal to	A>=B
<>	not equal to	A<>B
><	not equal to	A><B
=	equal to	A=B

String Operators:

Strings can be concatenated using the “+” operator. Strings can be copied to a string variable using the “=” operator. All other string manipulation is done with functions, see LEFT on [page 5-56](#), RIGHT on [page 5-82](#), MID on [page 5-61](#), LEN on [page 5-57](#), INSTR on [page 5-51](#), STR on [page 5-99](#) and VAL on [page 5-108](#).

Tabs:

Tabs are treated as spaces in TxBASIC.

Timing:

In standard BASIC dialects, there is little need to pace a program (the sooner it's over, the happier you are!). In a logging / control application however, program timing is critical. This function is handled by the SLEEP command in the Tattletale. SLEEP puts the logger in a low-power mode for an integral number of 10mS steps from the wake-up of the previous SLEEP command. This not only provides the necessary timing, but also ensures that the logger is in a low-power mode during the interval.

Types:

TxBASiC supports long integers, strings and single precision floating point. There are both implicit and explicit conversion operators to convert numbers between integer and floating point formats and back again. There are functions to convert between strings and numeric values. Refer to [“TxBASiC Floating Point”](#) on page 4-46 for details.

Variables:

Variable names up to 32-characters long are allowed in TxBASiC. Variables intended to be floating point variables must have a ! suffix when they are first used. The ! is not part of the name, though, and does not need to be used in later references to this variable. Refer to [“TxBASiC Floating Point”](#) on page 4-46 for details. Likewise, string variable names must have \$ suffix on the first use. Each string variable always takes up 256 bytes of variable storage whether it is used or not.

White Space:

White space is stripped out (except in strings) before it is sent to the Tattletale. White space is necessary around command, variable and array names.

TxBASiC Structure

The TxBASiC program development path is shown in Figure 4-1. TxBASiC differs from interpreted BASIC in that it adds a tokenizing pass between editing the program and sending it to the Tattletale. This pass is transparent (it is part of the routine that sends the program to the Tattletale). It's fast too, taking only about 20 seconds for a big program on the smallest computer.

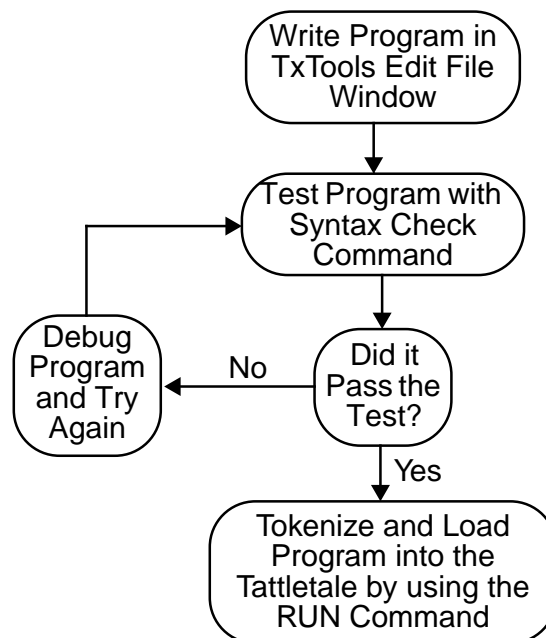


Figure 4-1: Flow Chart of the Development Path for a TxBASiC Program

The TxTools Program: Tokenizer & Program Loader

Before a program is sent to the Tattletale it is tokenized by the host computer (an Apple Macintosh or an IBM (or compatible) PC/AT computer). The tokenizer reads each command line and splits it into discrete, basic operations, and converts all expressions to their Reverse Polish equivalents. Each operation is defined by a token (or label) and a set of parameters. Once tokenized, the program is sent to the Tattletale where it can be run.

If a syntax error is found during tokenizing, the program is not loaded and the offending line is flagged so that the error can be corrected with TxTools.

Development programs are available for the Macintosh and IBM PC compatible computers. Both versions are explained in detail in [Section 3 - Operating the TxTools Program](#).

The Tattletale Program: Token Engine

The Tattletale's interpreter is in EPROM (EEPROM in the Model 5F, 5F-LCD, 6F and 8) and is distinctly different from the interpreter used by TTBASIC. TxBASIC can load a program and off-load a program from the Tattletale so it can be burned into EPROM (EEPROM on the Model 5F, 5F-LCD, 6F and 8), and off-load data.

Details of the Tattletale Program

The internal Tattletale program has two parts: a monitor and an interpreter (see Figure 4-2).

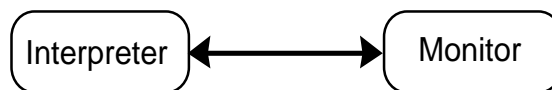


Figure 4-2: Tattle Program Parts

Interpreter:

The interpreter executes the TxBASIC program jumping to the Monitor on an error (unless redirected with the ONERR command), receiving a CTRL-C (unless redirected by the CBREAK command), or after executing a 'STOP' command (see Figure 4-3).

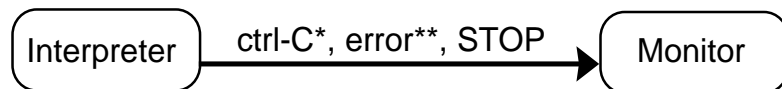


Figure 4-3: Tattletale Interpreter

* Only if not redirected by CBREAK

** Only if not redirected by ONERR

Monitor:

The Tattletale internal monitor has all the functions listed in Table 4-3. The commands shown are not user commands, they are commands the Tattletale uses internally.

Table 4-3: Tattletale Internal Monitor Commands

Internal Monitor Command	Command Function
CTRL-B	Boot the program in ROM to RAM, force a wrong checksum, and then start the interpreter. By forcing the wrong checksum, the program in RAM will be booted.
CTRL-R	Start interpreter on the program already in RAM after creating a valid checksum. The checksum is made of the TxBASiC program only.
CTRL-L	Load a tokenized program into RAM.
CTRL-E	Start XMODEM off-load of the operating system and the program area. In the Model 8, this command is used to burn your program into EEPROM.
CTRL-O	Start an XMODEM off-load of the datafile.
CTRL-V	Access the variables area.
CTRL-X (Model 8 only)	Exit TxBASiC and enter the Model 8 monitor. When prompted, type 'Y' (upper case) within 10 seconds.
CTRL-MINUS (Model 8 only)	Erase a user's TxBASiC program burned into EEPROM. When prompted, type 'Y' (upper case) within 10 seconds.

On Power-up:

On power-up the Tattletale always checks to see if there is a usable program in RAM by verifying the RAM checksum (see Figure 4-4). If the checksum is valid it will start the program; if not, it will boot the program in ROM (at the very least, TxBASiC will be in the ROM), and run it instead. In either case it will be running the interpreter.

NOTE: Any change made to the program code after start-up will make the program look invalid to the Tattletale.

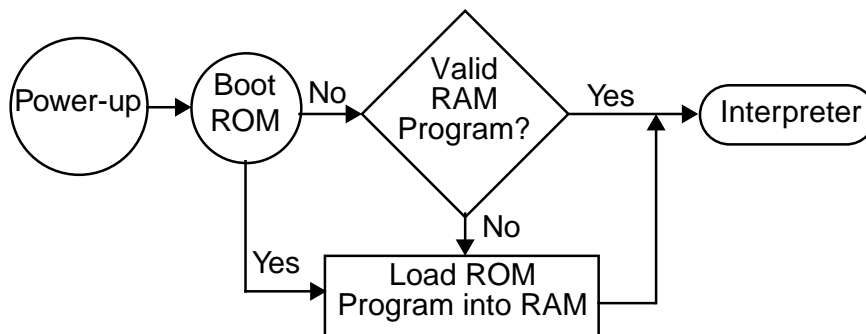


Figure 4-4: Tattletale Power-up Sequence

NOTE: The Model 8 *always* attempts to load the ROM program. This fails only if the EEPROM program has been corrupted.

Learning to Use TxBASIC

Flowcharts

To help visualize exactly what you want the Tattletale to do for you, some programmers find it easier to create a flowchart of the program. A flowchart will make writing the program much easier since all the flowchart symbols can be basically converted to a specific BASIC command and its structure. Table 4-4 shows all the symbols and their descriptions used for programming.

Table 4-4: Flowchart Symbols

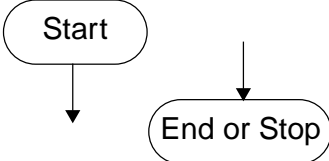
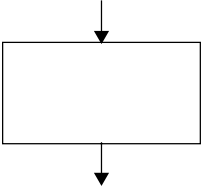
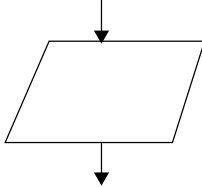
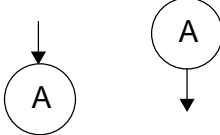
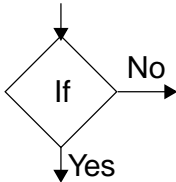
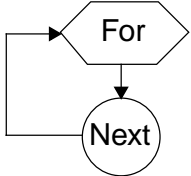
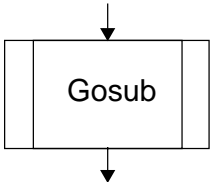
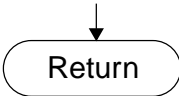
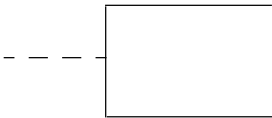
Symbol	Description
	Indicates the beginning or end of a program.
	Internal Process - Indicates some internal process of the computer (or Tattletale) such as reading a Thermistor for a temperature reading or storing the reading into a datafile.
	Input / Output - Indicates some communication between the computer (or Tattletale) and the user. For example: A prompt asking how many temperature readings to take would be an input statement and a temperature reading being printed on the screen would be an output statement.
	Connector - Used when the flowchart goes to another page or when drawing a line from one point to another would make the flowchart look sloppy or difficult to read.
	Decision - Indicates that a branch will occur in the program based on a result. The proper form for a decision branch is usually a Yes branch for a “true” statement and a No branch for a “false” statement.

Table 4-4: Flowchart Symbols (Continued)

Symbol	Description
	<p>For/Next Loop - Indicates that several commands will be repeated until a set number in the counter is met. Other commands like Repeat Until and While can also use these symbols.</p>
	<p>Sub-routine - Indicates that the program will branch to a different part of the program, execute the commands there until a RETURN command is encountered.</p>
	<p>End Sub-routine - This command directs the program back to the command line with the Gosub command.</p>
	<p>Annotation - This is not a program operation but a written note about the program or operation.</p>

Learning to Build a Data Logger, One Step at a Time

The following examples illustrate how to build a data logger starting with a simple temperature monitor and working up to a more complicated logger with provisions for readout.

NOTE: The Model 5, 5F, 5F-LCD, 6, 6F and 8 have no thermistor built-in and their results will be meaningless unless one is added. If you performed the installation procedure for the thermistor circuit in **Section - 2** the data read will be accurate and provide a more realistic tutorial.

Create a Flowchart for the Proposed Program

For our first program all we want the Tattletale to do is take a temperature reading and display it on the screen. Figure 4-5 shows the flow of our proposed program. Next to each part of the flowchart is a note showing what command will be used for that segment of the flowchart.

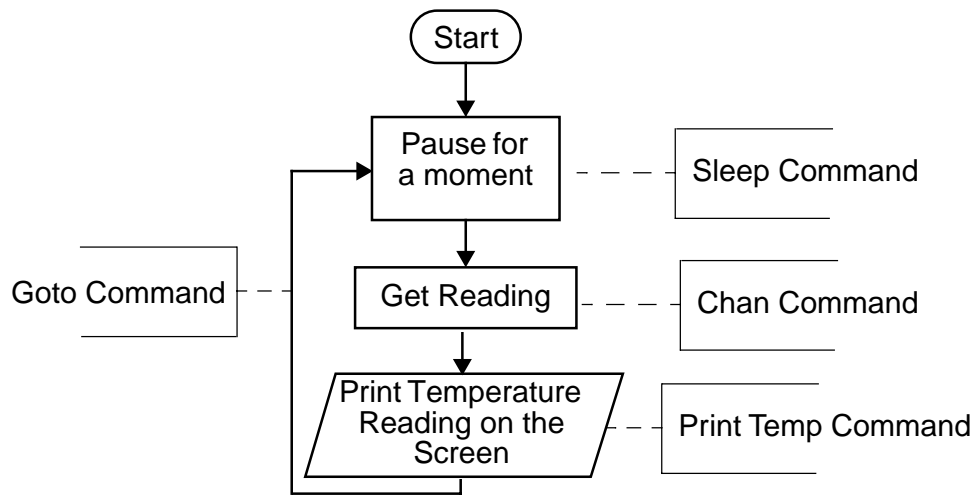


Figure 4-5: Tutorial Program Flowchart #1

Writing the Actual Program

Now that we have a flowchart for how the program will work, we can create the actual program code. The following procedure will show you exactly what to do to enter this program. **NOTE:** If you are using the Model 8, you will need to add **Model 800** as the first line of text in all your programs or the programs will not run.

NOTE: If you are using a Model 4, use CHAN(10) instead of CHAN(7) in the following examples.

1. Start the TxTools program.
2. Pull down the **“File”** menu and select **“New”**. A new untitled edit window will be displayed.
3. Pull down the **“File”** menu and select **“Save”**. Enter the name **“Tutorial”** and click the OK button.
4. Enter the following into the edit file window in TxTools:

```

label:  sleep 100
        print temp(chan(7))
        goto label
  
```

5. Pull down the **“Tattletale”** menu and select **“Run”**. Numbers similar to the following will be displayed:

```

#*2340
2340
2340
2340
  
```

6. Press Ctrl-C to stop the program. The # symbol prompt should be displayed.

This small program used five TxBASIC commands, SLEEP, PRINT, TEMP, CHAN and GOTO.

CHAN	The thermistor is hard wired to channel 7 on the Model 2A, 2B and 4A. All other Models (except the Model 4, which has a thermistor on channel 10) use channel 7 in the installation procedure in Section - 2 . The function CHAN(n) returns the converted input of channel 'n'.
GOTO	Forces the program to always branch to the defined label. Since line numbers are not used, you need to set up a label on the line to branch to.
PRINT	In this program, the Print command is used to display the data being read by the TEMP and CHAN commands on the screen.
TEMP	The thermistor is in a divider circuit with a 10K resistor. TxBASIC has a convenient function, TEMP, which converts the output of the converter to a temperature in hundredths of degrees C.
SLEEP	SLEEP places the Tattletale in a dormant mode until the specified time delay is complete (in our program it's 100mS).
*	The SLEEP command actually measures the time interval starting from the end of the previous SLEEP command in the same program. If this is the first time sleep was used in the program, and it was not used as SLEEP 0, it will print an "*" in front of the first line of the output text. If SLEEP 0 is used before any other SLEEP line it would be recognized as a starting marker, so the "*" in the first printed line would be suppressed.

Getting the Decimal Point Right

By modifying the program, we can print Celsius degrees as real numbers. With the following modifications, we will break the temperature into two parts. The new program will follow the flowchart shown in Figure 4-6.

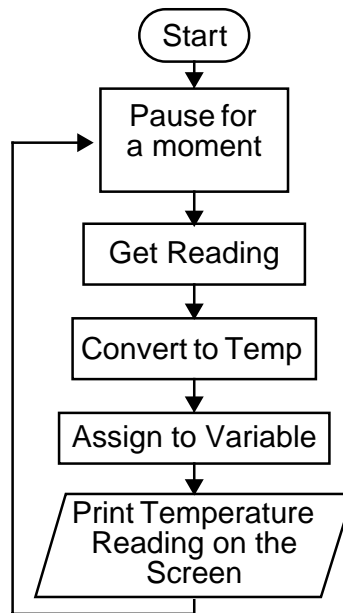


Figure 4-6: Tutorial Program Flowchart #2

1. Pull down the **“Windows”** menu and select **“Next”** (IBM PC) or **“Locate Tutorial”** (Macintosh) to switch from the terminal window to the edit file window.
2. Modify your program in the TxTools edit file window so that it reads:

```

label:  sleep 100
        let tempValue = temp(chan(7))
        print tempValue/100,!,#02,tempValue% 100
        goto label
  
```

3. Pull down the **“Tattletale”** menu and select **“Run”**. The following will be displayed:

```

#*23.40
23.40
23.40
23.40
  
```

4. Press Ctrl-C to stop the program. The # symbol prompt should be displayed. In the modified lines of code, we used three new commands and formatters: LET, #02 and the % sign.

LET	Assigns a value to a variable (in our program the actual temperature being read is being assigned to the variable “tempValue”. The Let command is optional in an assignment operation (tempValue is assigned the value of temp(chan(7))).
------------	---

#02	Specifies the format of the number to use. The '0' says fill with leading zeros, and the '2' tells the number of columns to use.
%	Returns the remainder of a division (% is the modulo operator in TxBASIC).
spaces	Ignored except in the middle of a number (1 9 4 5 is not recognized as 1945) and in the middle of a name (t e m p is not temp). You should include spaces around command, variable and label names.

Now Lets Display the Temperature in Fahrenheit

By modifying the program again, we will convert the temperature to Fahrenheit. We will also modify the program to use the floating point function of TxBASIC. The new program will follow the flowchart shown in Figure 4-7.

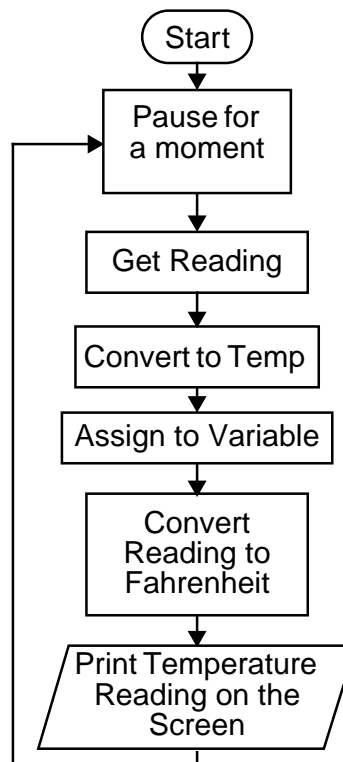


Figure 4-7: Tutorial Program Flowchart #3

1. Modify your program in the TxTools edit file window so that it reads:

```

label:  sleep 100
        let tempValue! = temp(chan(7))/100.
        tempValue = tempValue * 9. / 5. + 32
        print #.2f, tempValue
        goto label
  
```


- Pull down the “**Tattletale**” menu and select “**Run**”. Numbers similar to the following will be displayed:

```
*74.12
74.12
74.12
74.12
```

- Press Ctrl-C to stop the program. The # symbol prompt should be displayed.

!	The ! means this is a floating point variable.
.	The period means this is a floating point constant.
and &	The “ ” and “&” operators perform digital logic operations on TxBASIC numeric variables and constants. “&” performs the logical “AND” operation. “ ” performs the logical “OR” operation. These operators work in a bit-by-bit (or bit-wise) manner. That is, they don’t treat the values as numbers but as a collection of bits that can take on the value of 0 or 1.

Store Data and then Print to the Screen

Up to now we have only been reading and displaying the temperature continuously. With three more new commands we can have the program store 20 readings in the datafile on the Tattletale and then print out all the readings at once, neatly in horizontal rows. The new program will follow the flowchart shown in Figure 4-8.

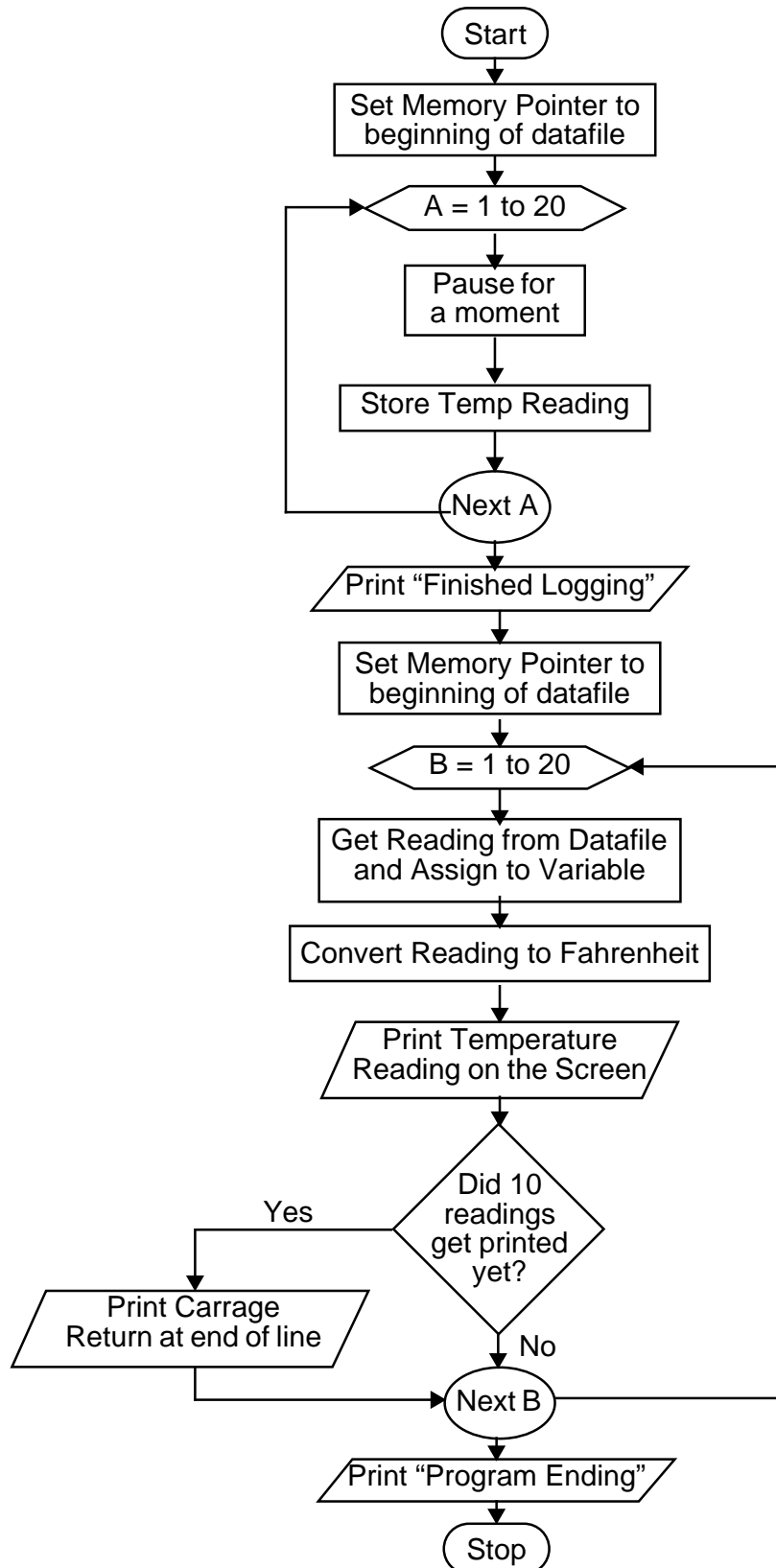


Figure 4-8: Tutorial Program Flowchart #4

1. Modify your program in the TxTools edit file window so that it reads:

```

dfPoint=0
for A = 1 TO 20
  sleep 10
  store dfPoint,#2,chan(7)
next A
print "Finished logging"
dfPoint = 0
for B = 1 TO 20
  tempValue! = temp(get(dfPoint,#2))/100.
  tempValue = tempValue * 9. / 5. + 32.
  print #6.2f, tempValue;
  if B % 10 = 0 print
next B
print "Program Ending"
stop

```

2. Pull down the “**Tattletale**” menu and select “**Run**”. The following will be displayed:

```

*Finished logging
74.12 74.12 74.12 74.12 74.12 74.12 74.12 74.12 74.12 74.12
74.12 74.12 74.12 74.12 74.12 74.12 74.12 74.12 74.12 74.12

```

Here the logging and printing parts are separated. The program stored the raw 16-bit data as it came in, and then printed out the stored results. In the last program, we added a data storage area. We call this data storage area the 'datafile'.

Datafile	<p>This is your data storage area. Each location in the datafile can hold one byte (a value between 0 and 255). The Tattletales have different sized datafiles; some can be extended to over 2 megabytes using add-on boards. The datafile is accessed through pointers, variables that have a value equal to the address of the location stored to. The storing variable is incremented after each byte is stored. In the above example, “dfPoint=0” is used as the pointer and is set to zero by the first line, making it point to the first location in the datafile.</p> <p>If multiple bytes (like strings of characters) are stored, the command will make sure the pointer is incremented past the last byte stored.</p>
-----------------	--

STORE	<p>The STORE command stores the variable tempValue using “dfPoint” as a pointer into the datafile. “dfPoint” is incremented after the storage operation. When “dfPoint” is incremented past the last storage location of the Tattletale, the STORE command causes an error which stops the routine.</p> <p>The default mode of the STORE command is to store the least significant byte only. In this case we need to store two bytes so we use the #2 specifier in the STORE and GET commands; if we wanted to store all four bytes of the variable we could have used a #4 specifier. Only #1, #2, #4 or no specifier (acts like #1) are valid.</p>
IF	<p>The IF statement allows you to branch on a condition. If the condition following the 'IF' is true, the rest of the line is executed; otherwise, the rest of the line is skipped.</p>
;	<p>The semicolon at the end of second PRINT suppressed the carriage return, and the next line put a carriage return after every tenth line.</p>

Turn-on Delay

The following program waits until I/O pin D0 goes high at which point the program will take 20 temperature readings and display them on the screen.

1. Write the following program in a new TxTools edit file window (you will need to use the MODEL command at the beginning of the program for this to work correctly):

```

sleep 0
print "Pulse pin D0 high to start"
wait: sleep 10 : if pin (0) = 0 goto wait
print "Starting to log"
dfPoint = 0
for A=1 to 20
    sleep 10
    store dfPoint,#2,chan(7)
next A
print "Finished logging"
dfPoint = 0
for B = 1 to 20
    tempValue = temp(get(dfPoint,#2))
    tempValue = tempValue * 9 / 5 + 3200
    print #4, tempValue / 100, ",", #02, tempValue % 100;
    if B % 10 = 0 print
next B

```

2. Pull down the “**Tattletale**” menu and select “**Run**”. The following will be displayed:

Pulse pin D0 high to start
 Starting to log
 Finished logging

```
76.62 76.62 76.62 76.62 76.62 76.62 76.62 76.62 76.62 76.62
76.62 76.65 76.65 76.65 76.65 76.65 76.65 76.65 76.65 76.65
```

Here, we simply programmed the Tattletale not to start logging until I/O pin 1 has gone high. There are several new things to learn here .

PIN	PIN returns a value of 0 if the pin is off, and non-0 if at the pin is on. Refer to the PIN command on page 5-71 for additional information.
//	Comment delimiter - The Tattletale ignores the rest of the line following the double slash //.
INPUT	This command allows you to assign values to variables as the program is executed. Notice there is no punctuation between the string constant and the variable name. Space is allowed for neatness NOTE: The input command can only be used to input numbers, not text strings
FOR	This command, along with NEXT, forms a powerful looping structure in TxBASIC. The loop starts with the assignment of the first specified value (1) to A and executing all the code up to the NEXT command. At the NEXT command, the variable (A) is incremented and the program goes back to the FOR line. The new value of the variable is then compared with the value after the TO; if it is less, the intervening code between the FOR and the NEXT is executed again, if not, execution is passed to the line following the NEXT statement.

Adding Bells and Whistles

One of the most common uses of the Tattletale is as a data logger, measuring analog channels and storing them for later readout. This example logging program allows the user to select the number of channels and logging interval. The layout of this program is a very good example of a well written program. By using many blank lines, and comment lines, additional clarity is achieved for the whole program.

1. Write the following program in a new TxTools edit file window:

```

//***** SAMPLE DATA LOGGING PROGRAM *****/

//**** First set up variable parameters****

InputData:    print
               input "Enter channel number to read (0-10): "numberOfChan
               print
               input "Enter delay time (mS): "timeInterval
               print
               input "Enter number of readings to take: "numberOfReadings
               print

//****INITIALIZATION****

               dfPoint = 0                // set memory pointer to zero
               onerr exit                 // quit when memory overflows
               sleep 0                   // initialize interval

//**** LOGGING ****

               for A = 1 TO numberOfReadings
                 sleep timeInterval
                 store dfPoint,#2,chan(numberOfChan)
               next A
               print
               print "Finished logging data"
               print
               input "Display the data on the screen now? (1 = Yes): "PrintNow
               print
               if PrintNow = 1 goto StartPrint
               goto exit

//**** PRINT OUT THE DATA *****/

StartPrint:    dfPoint = 0
               for B = 1 TO numberOfReadings
                 tempValue! = temp(get(dfPoint,#2))/100.
                 tempValue = tempValue * 9. / 5. + 32.
                 print #6.2f, tempValue;
                 if B % 10 = 0 print
               next B
               print
               print

//**** ENDING ****

               input "Would you like to take more readings? (1 = Yes): "more
               if more = 1 goto InputData
               print
exit:          print "Your data will be saved until the Tattletale is turned off."
               stop

```

2. Pull down the “**Tattletale**” menu and select “**Run**”. The following will be displayed:

Enter channel number to read (0-10): 7

Enter delay time (mS): 100

Enter number of readings to take: 50

Finished logging data

Display the data on the screen now? (1 = Yes): 1

```
75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21
75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21
75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21
75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21
75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21 75.21
```

Would you like to take more readings? (1 = Yes): 2

Your data will be saved until the Tattletale is turned off.

#

This concludes the programming examples using the Tattletale to record temperature. As you can see, it is easy to create a very intuitive program for the Tattletale. There are a couple of programming examples remaining which will show several additional commands.

blank line	Blank lines are permissible in TxBASIC. Use them to make your program more readable.
:	A colon permits multiple commands on a line.
ONERR	When ONERR is used, the program responds to an error instead of printing the error message 'How?'. The ONERR command forces the error to cause a jump to line labeled 'exit' instead. Should include a warning and point them to the ONERR command reference. NOTE: When an error occurs, any parameters on the stack are lost. Refer to the ONERR command on page 5-65 for additional information.
STOP	This command ends program execution. It really isn't needed here because it is the last program line which halts execution anyway.

Typing Test Program

This program will introduce a few new commands for storing text characters in the data storage area on the Tattletale.

1. Write the following program in a new TxTools edit file window:

```
// ***** Typing test *****

print "Type A through Z, then carriage return."
print
dfPoint = 0, timeValue = ?
itext dfPoint
timeValue = ? - timeValue
print
print "That took you ";
print timeValue / 100, ".", #02, timeValue % 100, " Seconds"
dfPoint = 0, numberOfErrors = 0
for A = &H41 to &H41+25
  if A <> get(dfPoint) numberOfErrors = numberOfErrors + 1
next A
print numberOfErrors, " Errors"
dfpoint = 0
print "You typed ";
otext dfPoint
```

2. Pull down the "**Tattletale**" menu and select "**Run**". The following will be displayed:

```
Type A through Z, then carriage return

ABCDEFGHIJKLMNOPQRSTUVWXYZ

That took you 8.59 Seconds
0 Errors
You typed ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

,	All lines beginning with an apostrophe are ignored as they are entered.
ITEXT	ITEXT has several forms. This one brings a string of characters ending with a CR to the datafile, updating the pointer when finished.
&H	This is a prefix that means that a hexadecimal number follows. For example &HFF = 255, and &HFFFF = 65535.
GET	GET retrieves a byte from the datafile, incrementing the pointer variable after it is done. GET has byte, word and double word forms.
OTEXT	OTEXT sends out a string ending in a carriage return.

Reaction time test

This program tests your reflexes and introduces a few more important TxBASiC operations. This example program will show you the remainder of our commands for the TxBASiC tutorial. After this you should refer to the detailed TxBASiC commands which are in alphabetical order in [Section 5 - TxBASiC Command Reference](#).

NOTE: You will need to add a high impedance speaker between pin 12 and ground, and a switch between pin 4 and ground.

Ten seconds after the program starts, the speaker will emit a 150mS beep, and the TxBASiC program will wait until you close the switch and measure how quickly you react.

NOTE: Pin 12 can't drive much of a speaker directly; you may need to add a simple driver to help it along (or just listen hard for the beep!)

1. Write the following program in a new TxTools edit file window (you will need to use the MODEL command at the beginning of the program for this to work correctly):

```

sleep 0:sleep 1000
tone 300,300           // use Tone 300,30 for Model 8
timeValue = ?
wait: if pin(4) <> 0 goto wait
timeValue = ? - timeValue
print "Your reaction time was "
print timeValue / 100, ".",#02,timeValue % 100, " Seconds"

```

2. Pull down the "**Tattletale**" menu and select "**Run**". The following will be displayed:

```

Your reaction time was 0.09 Seconds
#

```

?	The question mark variable is incremented every 10mS. In this case it is used to measure the elapsed time.
TONE	<p>TONE presents a square wave with the specified period and number of cycles. The period is measured in units of 1.6276µSec; thus, the value 300 yields a frequency of $1/(300 * 1.6276\mu\text{Sec}) = 2048\text{Hz}$. Three hundred cycles of 2048Hz takes 0.146sec.</p> <p>The Model 8 period units are 1.0µSec. Count is in 10ms count. 300 = 3333Hz 30 = 0.3sec</p>
PIN	PIN measures the state of one of the I/O lines. We picked number 4 because it has a pull-up on all Tattletales.

#02	This is a format specifier. The '2' says to use two spaces, and the leading zero says fill with zeros instead of spaces. This form is important to remember as the print statement normally suppresses the leading zeros in a number which would lead it to print an odd looking number; in this case (0. 9 instead of 0.09)! We could specify #1 instead of #01 in the previous examples, because TxBASiC will print the zero if only one digit is asked for.
------------	--

Tattletale Error Messages

There are two different kinds of errors recognized by TxBASiC. The first type is caught in the tokenizer. These are usually editing or syntax errors that produce the "Parse Error" display. The second type of error is caught at run-time. These are usually logical errors. When an error is encountered, program execution is halted, the word "HOW" and the error code number are displayed followed by the line number of the offending token (refer to Figure 4-9). The token line number can be found in the Token list file when using the Create Commented List option in TxTools and running the Syntax Check command or the Run command. The "How" error code numbers and their causes are listed in Table 4-5.

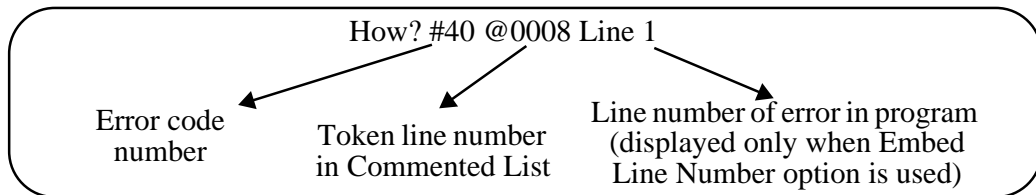


Figure 4-9: How? Error Line Display

Table 4-5: TxBASiC "HOW" Error Code Listing

#	Cause of Error	#	Cause of Error
1:	Not used in TxBASiC	28:	XSHAKE time-out > 65535
2:	Array variable index out of range	29:	Date/time input to STIME out of range
3:	Not used in TxBASiC	30:	Integer input to RTIME out of range
4:	STORE out of range of the datafile	31:	HYB internal > 65535
5:	GET out of range of the datafile	32:	SDO requested <1 or > 32 bits
6:	Integer divide by zero	33:	CALL address > 65535
7:	Integer multiply overflow	34:	Not used in TxBASiC
8:	Not used in TxBASiC	35:	Not used in TxBASiC
9:	Integer add or subtract overflow	36:	VSTORE/VGET index out of range
10:	Integer ABS argument = -2147483648	37:	UGET time-out > 65535

Table 4-5: TxBASIC “HOW” Error Code Listing (Continued)

#	Cause of Error	#	Cause of Error
11:	A-D channel not supported	38:	STIME out of range
12:	I/O pin not supported	39:	Not used in TxBASIC
13:	Not used in TxBASIC	40:	PRINT {x,y}; where y < x
14:	Input to TEMP out of range	41:	PRINT field width > 255
15:	Not used in TxBASIC	42:	INT or FIX result overflow
16:	SDI requested <1 or > 32 bits	43:	OFFLD x,y; where x > y
17:	COUNT time-out > 65535	44:	PIN number out of range
18:	Bad PERIOD argument	45:	Not used in TxBASIC
19:	Not used in TxBASIC	46:	Attempt to RUN a second background task
20:	Not used in TxBASIC	47:	Array index out of bounds
21:	Not used in TxBASIC	48:	DFSAVE, DFREAD out of bounds
22:	Datafile out of range (Off-load only)	49:	RATE 0 attempted
23:	OFFLD time-out > 65535	50:	ADLOOP address out of range
24:	SLEEP interval > 32767	51:	Stack running low
25:	USEND/UGET baud rate out of range	52:	Not used in TxBASIC
26:	ITEXT time-out > 65535	53:	Disk command: datafile wrong size
27:	TONE parameter out of range		

NOTE: For applications where these responses are undesirable, they can be replaced by a 'goto on error' response using the ONERR command.

Advanced use of TxBASIC

Dual Tasking

NOTE: Dual tasking is not available for the Model 8.

The most exciting feature added by TxBASIC is the ability to run a rigorously timed program in the background (perhaps collecting data at a set rate) while a foreground task shares the processor (perhaps checking for operator input).

General Guidelines for Background Tasks:

- Shouldn't use ITEXT, INPUT, TONE (Burst Mode), Long Count, Period or Offload in background.
- Must use Sleep or Stop to allow foreground to execute.

- Some commands will be affected. USEND and UGET in foreground will be adversely affected by the background task.
- The foreground task is defined as starting with the first line of the program.
- The background task can only be launched from the foreground task (using the RUN command as described on [page 5-84](#)).
- Once the background task begins, it only returns control to the foreground if it executes a STOP or a SLEEP.
 - a. If the background task executes a STOP, only the foreground task can start it again (or another background task).
 - b. If the background task executes a SLEEP, the foreground runs again but is interrupted every 10mS (or the rate chosen with the RATE command) to check if it's time to end the background SLEEP. Once the background SLEEP expires, the background task has full control.
- When a background task is running, the SLEEP timing for the foreground is no longer rigorous (it's easy to oversleep). So the * print is disabled for the foreground task - only while a background task is running. To distinguish between SLEEP commands, the background task prints a tilde '~' if it oversleeps instead of *.
- All TxBASIC variables are global. That is, they are available to both the foreground and background. They can be used to pass messages or flags between the two tasks. The example below shows how the foreground can signal the background to stop itself when the time is ready.
- Use the RUN command to launch a background task. Follow the RUN with the label of the routine you want to execute in the background. For example - 'RUN GetData' will cause the program starting at label 'GetData' to run in the background. Presumably, the routine starting at label 'GetData' has a SLEEP or STOP somewhere to pass control back to the foreground.

NOTE: The second task is NOT a GOSUB statement, do NOT use RETURN to end the background task.

- When a background task is started with RUN, it doesn't start immediately. A flag is set in TxBASIC that shows it is available to start. It waits in suspended mode until the next clock interrupt occurs (every 10mS or the rate chosen with the RATE command) which truly starts the background task.

The following is an example of a working dual tasking program:

```
print "Foreground prints + character, background prints message"
sleep 0: A = 0: B = 0           //reset the SLEEP count and flags
print "Foreground starts background #1"
run firstBG                   //start first background task
```

```

        for I = 0 to 50
        print "+"; : sleep 20           //print + symbols for 10 seconds
        next I
        A = 1                          //signal to background task to stop
wait1:  if A <> 0 goto wait1           //wait for background to acknowledge
        print : print "Foreground stops #1, starts #2"
        run secondBG                  //start second background task
        for I = 0 to 50
        print "+"; : sleep 20           //print + symbols for 10 seconds
        next I
        B = 1                          //signal second task to stop
wait2:  if B <> 0 goto wait2           //wait for background to acknowledge
        print : print "All background activity done"
        stop                          //stop foreground after b.g. stopped

firstBG: sleep 0                       //Start of Background Task 1
loop1:  if A <> 0 goto exit1           //foreground sets A to 1 to stop
        print "Executing task 1"
        sleep 200                      //message every two seconds
        goto loop1
exit1:  A = 0                          //acknowledge receipt of message
        stop                          //end background task 1

secondBG: sleep 0                     //Start of Background Task 2
loop2:  if B <> 0 goto exit2           //foreground sets B = 1 to stop
        print "Executing task 2"
        sleep 100                      //this message every second
        goto loop2
exit2:  B = 0                          //acknowledge receipt of message
        stop                          //end background task 2

```

The program has two possible background programs. One labeled **firstBG** and the other labeled **secondBG**. The foreground program starts the background task, works for 10 seconds and then sends a signal to the background to stop itself. This is important. There is no way for the foreground to stop the background directly. This allows the background to clean up before it stops. Otherwise, it could be in the middle of an operation when a foreground task brought it to a halt with some operation only partially done.

Notice that the SLEEP command can be used in the foreground and background simultaneously. Separate SLEEP counters are reserved for each task. The background takes precedence, though, so foreground SLEEP timing is not rigorous if a background task is running. The oversleep * printout is disabled from the foreground task when a background task is running. The background prints a ~ if it oversleeps (to distinguish it from the foreground).

This example shows one simple method of signaling between background and foreground. The 'A' variable is originally set to 0. When the time comes for the background to stop, the foreground sets A to 1. Since this could have been set while the background is SLEEPing, the foreground must wait for an acknowledgment that the background has received the signal and is ending. This is done by the background resetting A to 0. If the foreground just went on and tried to start the second background task, and the first background task was still running, an error (HOW error #46) would be signaled.

Another example of a dual tasking program (you will need to use the MODEL command at the beginning of the program for this to work correctly):

```

dfptr = 0, flag =0
rate 6
run background
loop:  print "A 300 Hz square wave is running on pin 2"
       print "Type any character to stop it."
       itext dfptr, 0
       if dfptr = 0 goto loop
       flag = 1
       stop

background:  sleep 1
            pset 2
            sleep 1
            pclr 2
            if flag = 0 goto background
            stop

```

This example is somewhat easier since there is only one background task. Watch the square wave with a scope.

TxBASIC Assembly Language

NOTE: This is not available for the Model 8 at this time.

The TxBASIC tokenizer (running on the host computer) has a built-in assembler. The tokenizer switches from generating tokens to assembling when it encounters the ASM command and switches back to tokenizing when it encounters the END command. This assembler allows the use of named labels and it can access TxBASIC variables by name.

General TxBASIC Assembly Language Information

Labels, Assembler:

Labels can be used in the assembly code for flow control and to define local variables. *Labels must start in the first column.* Labels can be up to 32 characters long and must begin with a letter or an underscore (_). The only valid characters in a label are upper and lower case characters, the numbers and underscore. The label name must be terminated with a colon (when the label is defined) but this is not necessary in the assembler. Assembly labels are accessible to TxBASIC via the Call command.

TxBASiC labels are not accessible to the assembly code (although TxBASiC variables are).

Assembler Opcodes:

The TxBASiC assembler recognizes all of the opcodes defined in the Hitachi 6301/6303 manuals except for BCLR, BSET, BTGL, BTST, BHS and BLO (which are just pseudonyms for the AIM, OIM, EIM, TIM, BCC and BCS opcodes respectively). Opcodes must have at least one character of whitespace (space character or tab) in front of them on the line OR a label terminated with a colon.

Two forms of ASM:

Assembly routines can be either executed in-line with TxBASiC commands or accessed as subroutines from TxBASiC using the CALL command depending on the argument that follows ASM.

In-line assembly (ASM \$):

The assembler (built in to the tokenizer) assembles from the line after the ASM \$ until it detects the END command. The interpreter will switch from interpreting tokens to executing the assembly code when it reaches the ASM\$ statement, and go back to interpreting tokens when it reaches the END statement. No RTS, RTI or other special ending command is needed before the END statement.

Example 1 shifts each bit of a TxBASiC variable one bit to the left with the most significant bit rotated around to the least significant bit position. Notice how the TxBASiC variables are handled. The name ToBeRotated points to the most significant byte of the variable.

To access the other three bytes of the variable:

To get this byte of variable:	Use this form of variable name:
Most significant byte	ToBeRotated
Next most significant byte	ToBeRotated+1
Third most significant byte	ToBeRotated+2
Least significant byte	ToBeRotated+3

Example 1:

```
input "Value to rotate: " ToBeRotated
input "Number of bits to rotate: " NumberShifts
print "value before rotation ",ToBeRotated
asm $ < nothing else on this line - not even comments! >
    ldab    NumberShifts+3      ; get number of shifts in B register
loop beq   leave              ; if number of shifts is zero, exit
    rol    ToBeRotated+3      ; ls byte, ms bit to carry, garbage into ls bit
```

```

rol    ToBeRotated+2    ; carry into ls bit, ms bit into carry
rol    ToBeRotated+1
rol    ToBeRotated      ; done except ls bit of ls byte is garbage
bcc    no_carry         ; branch if carry = 0
ldaa   ToBeRotated+3   ; get here if carry bit set so
oraa   #1               ; set the ls bit
bra    endloop
no_carry
ldaa   ToBeRotated+3   ; carry bit (from ms bit) is 0 so
anda   #&HFE          ; clear the ls bit
endloop
staa   ToBeRotated+3   ; restore the ls byte
decb                   ; count one bit shift completed
bra    loop
leave end    < on this line, 'leave' is a label and 'end' is a command >
print "value after rotation ",ToBeRotated

```

Figure 4-10 shows the operation of the assembly routine for one pass around the loop. Each TxBASIC variable is 32 bits wide. The figure shows the four most significant and four least significant bits of variable ToBeRotated. The value in variable NumberShifts tells how many times to do this operation. In the assembly section, everything after a semicolon and up to the end of the line, is considered a comment. The assembler does not recognize TxBASIC comments or REM commands.

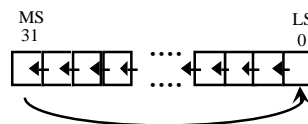


Figure 4-10: One Pass Through an Assembly Routine

The first form of the ASM command provides no way to initialize the A, B or X registers before entering the assembly code. The second form of ASM does, though.

Assembly to an address (ASM <address>)

When the interpreter reaches this point in the program, it DOES NOT EXECUTE THE ASSEMBLY CODE. It copies the assembly code to the address specified with the ASM command. Then it continues executing the tokens after the assembly code. To execute the assembly routine, you must use the CALL command from TxBASIC.

NOTE: An RTS instruction is automatically appended to any code you write. This is fine if you're writing normal assembly subroutines but you cannot use this method to write data to a RAM address. Use the POKE command instead.

The area between 118H and 13FH (General User Area 1 - 40 bytes) and the area between 74C0H and 75FFH (General User Area 2 - 320 bytes) are reserved for user assembly language code.

The TxBASiC 'CALL' command has this syntax:

CALL <x1>, <x2>, [<v>]

x1	The address of the assembly routine - this is the address following ASM - can also use an assembler label here.
x2	The state of the X, A and B registers to be upon entering the subroutine
v	An optional variable name to receive the state of the X, A and B registers when the subroutine returns to TxBASiC.

The translation of the **x2** and **v** arguments into the microprocessor's registers (Figure 4-11):

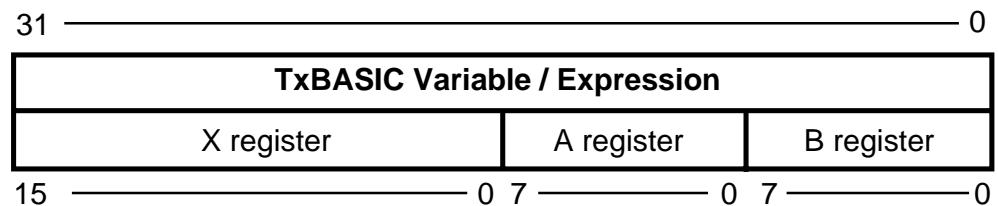


Figure 4-11: TxBASiC Variable / Expression

In the example below, the B register is filled with the ASCII value of a character to be sent out the UART. The X register is filled with the number of times to send this character (beware, zero in the X register will send it 65536 times!).

Example 2:

```
print "Start out in TxBASiC"
asm &H74C0 < assembly routine will be loaded at address 74C0H >
test tim    #20H,11H    ; test if transmit data register empty
    beq     test        ; branch back if not empty
    stab    13H         ; send character in B reg by storing in register 13H
    dex                      ; count down one character
    bne     test        ; if not zero, branch back for another
end
print "Asm code has been loaded at 74C0H"
sleep 0: sleep 50< must wait for PRINT statement output to finish >
call test, &H10041< X register= 1, A register= 0, B register= 41H = 'A' >
call test, &H20042< X register= 2, A register= 0, B register= 42H = 'B' >
call test, &H30043< X register= 3, A register= 0, B register= 43H = 'C' >
call test, &H1000D< X register= 1, A register= 0, B register= 0DH = CR >
call test, &H1000A< X register= 1, A register= 0, B register= 0AH = LF >
print "Finished"
```

Notice the line “sleep 0: sleep 50”. The PRINT command in TxBASIC sends all its output to a buffer. The characters in this buffer are sent out the UART in the background as fast as the baud rate will allow, but this is very slow compared to the speed of the computer. When the PRINT command is completed, TxBASIC continues on with the next command while the characters are still being sent out the UART. The assembly routine we wrote bypasses the UART buffer and will interfere with characters being automatically output from the buffer. So, we pause to let the buffer clear before using our routine.

Example 3: Pulse Width Measurement with a Tattletale

We have found substantial interest in measuring pulse widths with a Tattletale, and have written an assembly language patch to address this need. This patch is loaded into one of the General User Areas using ASM commands and then executed with a CALL statement.

The command works for pulse widths from 250µSec to 10 minutes, and uses pin D13 for its input.

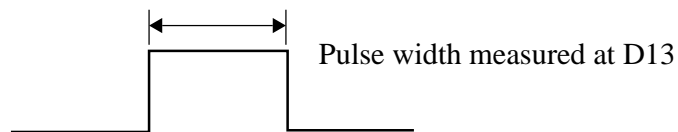


Figure 4-12: Pulse Width Measurement

To measure the width of a positive going pulse use the CALL command. The input parameter will be used as a time-out (in units of 10mS), and the measured width will be returned in the variable specified as the second parameter (zero returned if time-out before pulse over).

The address is &H74C0 for all Tattletale models. The time-out can be anything from 2 (20mS) to 65535 (about 10 minutes). The returned pulse width (in <variable>) is in units of 1/1.2288µSec, so a 10mS pulse will return 12288 in the variable.

```

REM first line of program must be DFSIZE command if used with Model 5.
  GOSUB PULSE      :REM load assembly language to USER RAM area #2
  INPUT "TIMEOUT = " A :REM input timeout value (10mS increments)
LOOP:  CALL &H74C0,A,B
      IF B=0 PRINT 'TIMED OUT '           :REM complain if timeout
      IF B<>0 PRINT B                     :REM otherwise print the pulse width
      GOTO LOOP

PULSE:  ASM &H74C0
;-----
;   PULSE WIDTH MEASUREMENT ASSEMBLY ROUTINE
;-----
  TLSB      EQU      43H      ;LSB OF TIME
  TOI       EQU      103H

```

ICI	EQU	109H
TCSR1	EQU	8H
ICRHI	EQU	0DH
FRCHI	EQU	9H

```

;-----
;
;
; FORM:      CALL X,<V>
;           X IS TIMEOUT (IN 1/100 OF A SECOND), 64K MAX
;           PULSE WITH RETURNED IN <V>, 0 IF TIMED OUT
;-----

        STD     R2
        LDAA   #7EH      ;JMP INSTRUCTION
        STAA   TOI
        STAA   ICI
        LDD   #INTICI    ;SET JUMP VECTOR
        STD   ICI+1
        LDD   #INTTOF   ;SET JUMP VECTOR
        STD   TOI+1
        LDD   #0
        STD   R3
        STD   R3+2
        CLR   R1        ;PASS COUNTER FOR ROUTINE BELOW
        LDAB  #2        ;SET POLARITY OF EDGE
        STAB  TCSR1
        LDAB  TCSR1    ;DEFEAT PENDING INTERRUPT
        LDD   ICRHI
        LDD   FRCHI
        LDAA  TLSB
        LDAB  #00010110B ;ENABLE CAPTURE, & TIMER OVERFLOW INTERRUPT
;
; WILL GO THROUGH THE EDGE ROUTINE TWICE, ONCE
; TO FIND THE FIRST EDGE, AND ONCE TO FIND THE
; SECOND EDGE. THE ROUTINE WILL EXIT IF TIMEOUT
; OCCURS. ON EXIT R4 WILL HOLD THE STARTING TIME
; AND R3 WILL HOLD THE ENDING TIME.
; AT START R2 HAS TIMEOUT,

PUL2   SEI
        STAB  TCSR1    ;RE-ENABLE THIS INTERRUPT
        CLI
        SLP
        CMPA  TLSB     ;QUIT ON TIMEOUT
        BEQ  PUL3
        LDAA  TLSB
        LDX  R2        ;TIMEOUT

```

```

    DEX
    BEQ    PULX    ;EXIT IF TIMED OUT
    STX    R2
PUL3  SEI                ;PROTECT FROM INTERRUPTS
    LDAA  R1            ;IF R1=1, SAVE R3 TO R4
    CMPA  #1
    BNE   PUL4
    LDX   R3
    STX   R4            ;MOVE COUNT TO R4 SO IT WILL
    LDX   R3+2          ;HAVE TIME OF FIRST EDGE
    STX   R4+2
    LDAB  #00010100B   ;CHANGE EDGE
    INC   R1            ;MAKE SURE SEE THIS ONLY ONCE
    BRA   PUL2
PUL4  CMPA  #3            ;PASS COUNT = 2, THEN DONE
    BNE   PUL2          ;ELSE BACK FOR MORE
    CLR   TCSR1        ;TURN OFF INTERRUPTS
    LDD   R3+2
    SUBD  R4+2          ;FORM R3-R4 TO GET WIDTH
    XGDX
    LDD   R3
    SBCB  R4+1
    SBCA  R4
    XGDX
    CLI
    RTS

PULX  CLR   TCSR1      ;TURN OFF INTERRUPTS
    LDX   #0
    CLRA
    CLRB
    RTS

INTICI LDAA  TCSR1      ;CLEAR INTERRUPT, AND DISABLE IT
    INC   R1            ;INCREMENT EDGE COUNTER
    ANDA  #0E0H        ;CHANGE EDGE
    STAA  TCSR1
    LDX   ICRHI        ;NEED TO READ THIS TOO
    STX   R3+2          ;SAVE VALUE IN R3
    ANDA  #20H         ;IF THIS BIT HIGH MIST INC R3
    BEQ   PULIN1       ;ELSE SKIP TO END
    LDX   R3+2
    CPX   #8000H
    BCC   PULIN1
PULIN2 LDX   R3
    INX
    STX   R3

```

```

        LDD    FRCHI    ;CLEAR TOI INTERRUPT
PULIN1 RTI

INTTOF  LDAA  TCSR1
        BRA   PULIN2

R1     DB    0
R2     DW    0
R3     DW    0,0
R4     DW    0,0
        END
        RETURN

```

Radix: Notice the new methods of defining the number base of constants. You have these options IN THE ASSEMBLER ONLY. For example, the constant 19 decimal can be defined as:

```

decimal: 19D or D'19 or 19           decimal is the default number base
hex:     13H or &H13 or H'13 or $13  &H works as in TxBASiC
octal:   23O or 23Q or Q'23 or @23
binary:  10011B or B'10011 or %10011

```

ASM Mnemonics:

The TxBASiC assembler uses the same opcode mnemonics that Hitachi uses in its 6301/6303 literature (Table 4-6). When you order a development kit for any Tattletale model, you receive the Hitachi information. All assembler mnemonics must have at least one character of whitespace (space character or tab) in front of them on the line or a label terminated with a colon.

Table 4-6: Assembly Language Addressing Modes

Single Byte:	The single byte commands, like NOP and RTS, can be preceded and/or followed by an arbitrary number of spaces and tabs.
Indirect:	<p>These commands use the 'X' register as a pointer. They have the form:</p> <p>MNEMONIC <expr>,X</p> <p>where <expr> is a positive offset and must evaluate to a number in the range of 0 to 255, and the 'X' (upper or lower case) must follow the comma without any intervening space. Examples:</p> <p>LDAA 0,X ; load A reg with byte pointed to by X</p> <p>STAA 7*8-4,X ; store A reg at address contained in X + 52</p>

Table 4-6: Assembly Language Addressing Modes

Extended:	<p>Extended commands address the full 64K memory. They have the form:</p> <p>MNEMONIC <expr></p> <p>If <expr> evaluates to a number outside the range of 0 to 65535, the assembler (in the tokenizer) will print an error message. Examples:</p> <p>LSR H'74C0 ; logical shift right of the byte at 74C0H ROL 118H ; rotate byte at 118H left one bit through carry</p>
Extended /Direct:	<p>Many commands have shortened versions (called 'direct' by Motorola) for addressing page zero of memory (address < 256), as well as full length versions for addressing all of memory. The 'direct' version of the command takes two bytes instead of three. The assembler first evaluates the <expr>. If the result is in the range 0 to 255, the assembler uses the direct form; otherwise, the assembler uses the extended form. Examples:</p> <p>SUBB 10 ; direct, subtract byte at addr 10 from B reg SUBB 256 ; extended, subtract byte at address 256 from B reg</p> <p>CMPA 255 ; direct, compare A reg with byte at addr 255 LDAA 7*128 ; extended, load A reg with byte at address 896</p>
Relative:	<p>The branch commands form a displacement from the address of the byte immediately following the branch command to the address that is to be branched to. In the form used by the assembler, the address is specified, not the displacement. The assembler will then calculate the displacement. You can use either a label or an absolute number to specify the address. Examples:</p> <p>BNE _NoMore ; branch if not equal to label '_NoMore' BRA 74C0H ; branch always to address 74C0H</p> <p>Relative branches must be within +127 and -128 of the start of the next instruction. If the displacement does not fall within this range, a tokenizer error will result.</p>

Table 4-6: Assembly Language Addressing Modes

Immediate:	<p>An immediate command deals with a value instead of an address. The form of the immediate commands is shown below:</p> <p>MNEMONIC #<expr></p> <p>There must be white space (tabs or spaces) between the MNEMONIC and the '#'. There can be white space between the '#' and the <expr>. The assembler checks <expr> and makes sure it is in range (0 to 255 for some commands, 0 to 65535 for others). Examples:</p> <p>LDAA #&H23 ; load A reg with 35 (23H) LDD #&H1234 ; load A/B registers with 4660 (1234H) SUBD #7*16+&H4000 ; subtract 16496 (4070H) from A/B registers</p>
AIM Type:	<p>The commands AIM, OIM, EIM and TIM have both direct (address < 256) and indirect forms as shown below</p> <p>MNEMONIC <expr1>,<expr2> ; direct MNEMONIC <expr1>,<expr2>,X ; indirect</p> <p><expr1> specifies the masking byte, and <expr2> specifies the page zero address (for the direct command), or the offset (for the indirect command). In the indirect command form, the effective address is the X register's value plus the unsigned offset <expr1>.</p> <p>AIM H'0F,H'BB ; direct, AND byte at addr BBH with 0FH OIM 23,&HFF,X ; indirect, OR byte at X + 255 with 23 EIM 4*8,260-8 ; direct, XOR byte at 252 with 32</p>

Summary of TxBASiC Assembler Directives

The assembler makes some directives and pseudo-ops available. Here is a brief listing of the directives in the TxBASiC assembler. The second section goes into more detail and provides examples of how the directives are used.

The TxBASiC assembler directives are grouped into four categories: source control, data declaration, symbol declaration and location control. The source control directives tell the assembler where its source code starts and ends. The data declarations allow you to initialize an area of memory from an expression or string. The symbol declaration directive allows you to assign a numeric value to a symbol name that can be used anywhere a constant is expected. The location controls allow you to set or modify the location counter (the address to which code is assembled). Notice that the TxBASiC command ASM is in two categories.

Source Control	
ASM	Start assembly source (not detailed here)
END	End assembly and return to TxBASiC
Data Declarations	
DATA	Declare initialized data (IEEE)
DB	"Define Byte" (Intel)
FCB	"Form Constant Byte" (Motorola)
DW	"Define Word" (Intel)
FDB	"Form Double Byte" (Motorola)
FCC	"Form Constant Characters" (Motorola)
Symbol Declarations	
EQU	Assign value to symbol
Location Control	
ALIGN	Force location counter alignment
ASM	Set the location counter (not detailed here)
RES	Reserve uninitialized data
DS	"Define Storage" (Intel)
RMB	"Reserve Memory Block" (Motorola)

Details of the TXBASiC Assembler Directives

END

Syntax:

```
[label] end    [; comments]
```

Description:

The end directive instructs the assembler to stop reading the current source file and return control to the TxBASiC tokenizer.

Example:

```
asm $
    ldaa    #12
    ldx     #H'86
    staa    0,x
    end          ; Configuration Byte set
```

DATA, DB, FCB, DW, FDB, FCC

Syntax:

```
[label] data[.size] expr | "string" [,expr | "string"...] [; comments]
[label] db      expr | "string" [,expr | "string"...] [; comments]
[label] fcb     expr | "string" [,expr | "string"...] [; comments]
[label] dw      expr | "string" [,expr | "string"...] [; comments]
[label] fdb     expr | "string" [,expr | "string"...] [; comments]
[label] fcc     <delim> "string" <delim> [; comments]
```

Size Specifiers (for 'data' directive only)

```
.b .B      Byte
.s .S      Short (2 Bytes)
.w .W      Word (2 Bytes)
.l .L      Long (4 Bytes)
.q .Q      Quad (8 Bytes)
```


Description:

The various data declaration directives instruct the assembler to generate initialized data from expressions and strings. The size of the data generated for each expression is determined either explicitly with "dot-size" suffixing as in the data directive, or implicitly by the directive name. Expression values are truncated to fit into objects of the specified size. The data directive and the "dot-size" suffixes shown in the table conform to the guidelines of the IEEE-694 standard for microprocessor assembly language. The fcb (Form Constant Byte), fdb (Form Double Byte), and fcc (Form Constant Characters) are provided as a convenience to programmers more comfortable with Motorola pseudo-ops, while the db (Define Byte), and dw (Define Word) are available for programmers preferring Intel mnemonics. With the exception of the fcc directive, all of the data directives accept any combination of strings and expressions. Strings used with these directives must be enclosed inside double-quotes. Use two consecutive double-quotes to generate one double-quote character in the output data.

The fcc directive generates data from string expressions bracketed by the first character encountered in the operand field. Use two consecutive occurrences of the delimiting character to generate one instance of the character in the output data.

Examples:

```

_data: data  "sample string"
      data  "several", "different", "strings"
      data  "several", 0, "terminated", 0, "strings", 0
      data  "sample string with quote (\"" character"
      data  "string with byte expression", 10
;
      data  0,1,2,3          ; yields: 00 01 02 03
      data.b 0,1,2,3        ; yields: 00 01 02 03
      data.w 0,1,2,3        ; yields: 00 00 00 01 00 02 00 03
      data.l 0,1            ; yields: 00 00 00 00 00 00 00 01
      data.q 0              ; yields: 00 00 00 00 00 00 00 00
;
_db:   db    "string with ending bit 7 hig", 'h' + h'80
_fcb:  fcb   "string with embedded quote (\""
      db    'ab'          ; should just be h'61
      fcb   '"'          ; the single-quote character
;
_dw:   dw    0,1,2,3      ; yields: 00 00 00 01 00 02 00 03
      dw    'ab'          ; should be h'6162
_fdb:  fdb   """"        ; two single-quote characters
;
_fcc:  fcc   :string with colon (::) terminators:
      fcc   ~string with tilde (~~) terminators~
      fcc   /string with tilde (/) terminators/

```

Limits:

Maximum of 255 bytes generated from a single data statement.

EQU**Syntax:**

```
label equ expr [; comments]
```

Description

The equ directive permanently assigns a numeric value to a symbol which may be used anywhere a constant is expected. Both the label and operand fields are required.

Examples:

```
FALSE      equ    0
TRUE       equ    ~FALSE
DEBUG      equ    FALSE
DEL        equ    H'7F ; ASCII delete
CR         equ    H'0D ; carriage return
LF         equ    H'0A ; line feed
TAB        equ    H'09 ; horizontal tab
SP         equ    H'20 ; 'space' character
```

Remarks:

Values used in the assignment expression must not contain any forward references. The TxBASiC assembler does not adapt EQU to perform text substitution on arguments which do not evaluate to a number.

ALIGN**Syntax:**

```
[label]    align    expr    [; comments]
```

Description:

The align directive forces the location counter to align to a boundary that is an even multiple of the value specified by the expression in the operand field.

Examples:

```
_align:    align    2          ; align on word boundary
            data    "word"
            align    4          ; align on long boundary
            data    "long"
            align    256       ; align on page boundary
            data    "page"
            align    32768     ; align on enormous boundary
            data    "huge"
            align    1         ; back to byte boundary
            data    "byte"
```

Remarks:

Values in the align expression must not contain any forward references.

RES, DS, RMB**Syntax:**

```
[label]    res      expr    [; comments]
[label]    ds       expr    [; comments]
[label]    rmb     expr    [; comments]
```

Description:

The `res`, `ds`, and `rmb` directives define a block of uninitialized data equal in length to the value of the expression in the operand field multiplied by the optional size specifier. Blocks declared using the reserve directives affect only the location counter and do not generate any output data in the object file.

Examples:

```
asm    h'4000
_res:  res    h'100    ; should be 4000 - 40FF
      res.b   h'100    ; should be 4100 - 41FF
      res.s   h'100    ; should be 4200 - 43FF
      res.w   h'100    ; should be 4400 - 45FF
      res.l   h'100    ; should be 4600 - 49FF
      res.q   h'100    ; should be 4A00 - 51FF
```

Remarks:

Values used in reserve statements must not contain any forward references.

Assembly Language Subroutines

Fixed vector locations of useful assembly language routines

Name	Addr	Explanation
ANIN	FFCA	Make A-D conversion of channel specified in B register. Result is returned in A/B registers. Power must be applied to A-D converter first with call to CONON and removed after with CONOFF (X register preserved).
CONON	FFCD	Turn on power to A-D converter. No return value (B & X registers preserved).
CONOFF	FFD0	Turn off power to A-D converter. No return value (all registers preserved except Model 2A and 2A-32 only preserve A & X registers).
STRMEM	FFD3	Use X register to point to a four-byte value (like a Tx variable) containing datafile address to which to store byte in A register. The 4-byte value is incremented after the store (X register preserved).
GETMEM	FFD6	Use X register to point to a four-byte value (like a Tx variable) containing datafile address from which to get byte returned in A register. The 4-byte value is incremented after the read (X register preserved).
URTTST	FFD9	Returns value in A register showing if characters remain in the UART input buffer. If zero, no characters. If non-zero, characters in buffer (B & X registers preserved).
URTGET	FFDC	Returns next byte from hardware UART buffer in A register. If no byte ready, will wait forever (in SLEEP-like mode) (B & X registers preserved).
URTSND	FFDF	Sends character in A register out hardware UART. All registers preserved.
FLUSH	FFE2	Flushes both UART buffers. No return value (X register preserved).

ATOD	FFE5	Same as ANIN except CONON/CONOFF called automatically (X register preserved).
TSTOUT	FFC7	Returns value in A register showing if characters remain in the UART output buffer. If zero, no characters remain to be transmitted (B & X registers preserved).
CHIBUF	FFC4	Returns next character in hardware UART input buffer without removing it from buffer. If a character is available, returns it in A register with carry flag set. If no character available, returns with zero in A register and carry flag clear (B & X registers preserved).
ALTCLK	FFC1	Do NOT call this routine directly! It is an interrupt routine and handles a system clock using Timer1 instead of Timer2 (Tattletale reset default). Call Onset for information about using this. It allows Timer2 to be used as a baud rate generator making new baud rates available for UART.

NOTE: In both **ANIN** and **ATOD**, interrupts are disabled on entry and enabled on exit. Whether or not you have interrupts disabled before calling **ANIN** or **ATOD**, they will be enabled on exit!

Important Addresses in TxBASiC

These system variables may be accessed in your assembly routines if you are careful.

Time variables	Addresses	Notes
? variable	40H - 43H	least significant 4 bytes
? variable	44H	most significant byte
? array -?(0)	45H - 48H	second
? array -?(1)	49H - 4CH	minute
? array -?(2)	4DH - 50H	hour
? array -?(3)	51H - 54H	day
? array -?(4)	55H - 58H	month
? array -?(5)	59H - 5CH	year

Microprocessor port data direction registers:

Time variables	Addresses	Notes
Port 2 DDR copy	69H	pins D8 - D13 and UART send/rcv
Port 5 DDR copy	6AH	pins D0 - D7
Port 6 DDR copy	6BH	pins D14 - D16 (Model 5 only)

Other Variables

CONFIG	86H	The configuration byte (see Section 6 - Hardware and Interface Specifications)
CCOUNT	93H	Count of Ctrl-C characters pending (see CBREAK on page 5-21)
XDATFL	9AH - 9DH	Number of bytes in datafile extension (Model 5 only)
CENAB	9EH	Ctrl-C enable (zero ignores Ctrl-C) (see CBREAK on page 5-21)
TRACE	B5H	Set to non-zero to trace execution by using Poke (&HB5, &H3C) and turn on the "Embed Line Numbers" option in the Tokenizer Flags sub-menu (see page 3-17 for IBM PC and page 3-37 for Macintosh). To turn off the trace use Poke (&HB5, 0).

TCRCPY	94H	Copy of the write-only TCR on the microprocessor
OVRSLP	B6H	Flags to show if SLEEP detected an oversleep condition. Bit 0 set if foreground, bit 1 set if background. Not cleared on RESET. You must clear this if you plan to check it later.

Interrupt jump vectors	Addresses	Normally used by
IRQ2 (pin 6)	100H - 102H	nothing
Timer 1 overflow (TOI)	103H - 105H	PERIOD command
Timer 1 output compare (OCI)	106H - 108H	TONE command
Timer 1 input capture (ICI)	109H - 10BH	PERIOD command
IRQ1 (pin 7)	10CH - 10EH	UGET command
Timer 2 count match (INTCMI)	10FH - 111H	? variable update (real-time clock)
Trap (illegal instruction)	112H - 114H	nothing
SWINT (software interrupt)	115H - 117H	SWI debugging printout

TxBASiC Memory Maps

TxBASiC Memory Maps

Shaded areas are ROM

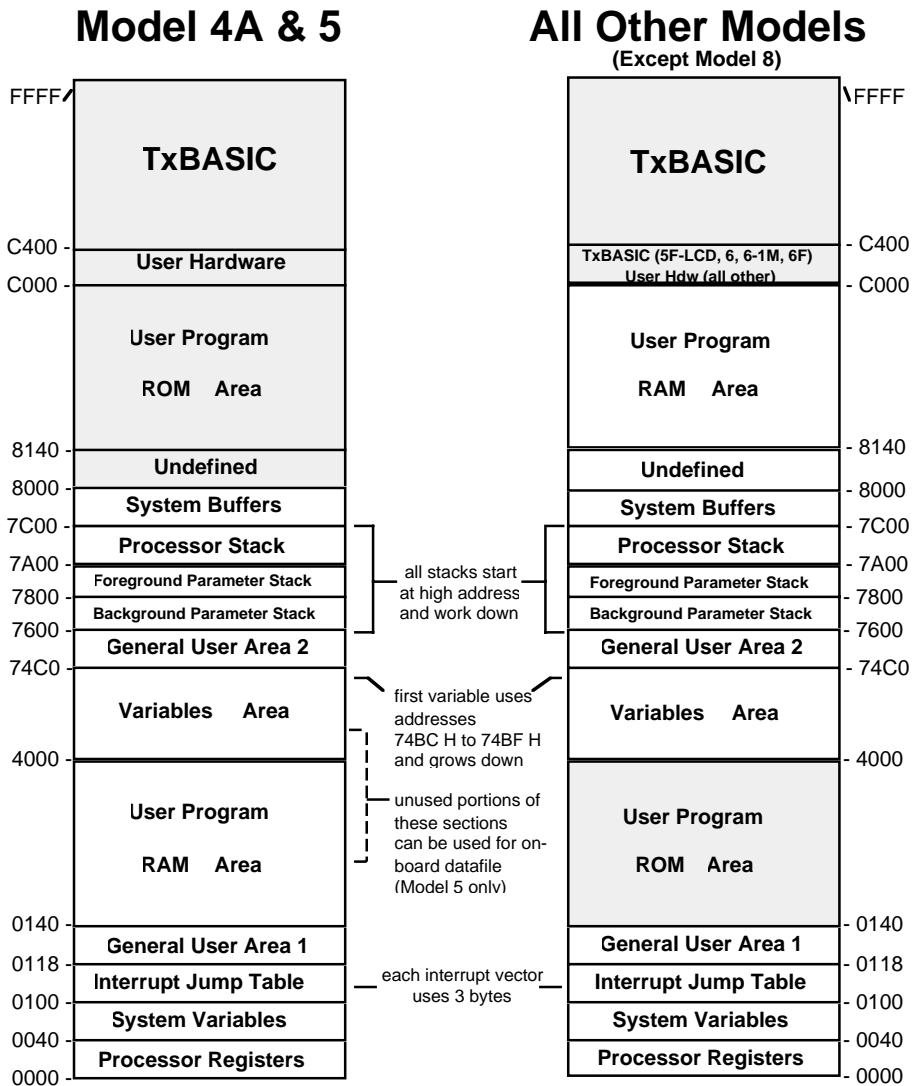


Figure 4-13: TxBASiC Memory Maps

NOTE: There is no memory map for TxBASiC on the Model 8. It's processor is a more sophisticated engine than the processor of the other Tattletales. Fixing the addresses of TxBASiC limits the power of the Model 8.

TxBASiC Variables

TxBASiC allows variables with names up to 32 characters long, and two special variables for dealing with time. The '?' variable is cleared when the Tattletale is reset. No other variable is modified. Variables can be either signed long integers in the range -2147483648 to +2147483647 or single precision floating point . All variables occupy four bytes.

The absolute addresses of variables are available through the VARPTR function. The range of single precision floating point is $\pm 1.175494E-38$ to $\pm 3.402823E+38$, zero and \pm Infinity. TxBASIC does not support string variables at this time.

Read-Only Variables

VERS

A read-only variable that gives the revision level of the logger's TxBASIC times one hundred. The command PRINT VERS in version 4.00 produces the string '400'.

MODEL

This read-only variable returns a three digit integer showing the model number of the Tattletale. The model numbers are shown in Table 4-7.

Table 4-7: TxBASIC Model Command Formats

Tattletale Model	TxBASIC Model Command Format
2A	model 210
2A-32	model 220
2B	model 230
2B-1M	model 240
3	model 300
4A	model 400
5	model 500
5F	model 510
5F-LCD	model 520
6	model 600
6-1M	model 610
6F	model 620
8	model 800

FPERR

A read-only variable that gives the types of floating point math errors that have occurred since last checked. Any reference to this variable clears it.

DFMAX

A read-only variable that gives the maximum datafile address for your Tattletale.

NOTE: Initialize your Model 4A, 5 and 8 expansion memory first to get an accurate value.

Arrays

An array is defined using the 'DIM' command. One and two dimension arrays are accepted. The ADLOOP array and ? array are predefined. TxBASIC allows floating

point arrays, too. Be sure to append an exclamation point ! to the array name when defining it with DIM.

? array & ? variable

To deal with time efficiently, TxBASIC provides the '?' variable and the '(0..5)' array. The '?' variable is a unique, five byte variable. Its value is incremented by one every 0.01 second. RTIME and STIME are TxBASIC commands that can convert the time in the '?' variable to seconds ... years. The array subscripts 0 through 5 correspond to 'real-time-clock' registers: seconds (0..59), minutes (0..59), hours (0..23), days (0..28, 29, 30, 31), months (1..12), and years ('80..'20). The RTIME and STIME commands are the only way to access all five bytes of the '?' variable. Any other use (like A = ? or PRINT ?) will only access the least significant four bytes of '?'. '?' is cleared to zero on power-up. If it is not reset or modified by the user, its least significant four bytes will overflow after about 200 days. See RTIME on [page 5-83](#) and STIME on [page 5-95](#).

TxBASIC Floating Point

The floating point format used is based on the IEEE 754 single precision floating point standard. The range of representable values is $\pm 1.175494E-38$ to $\pm 3.402823E+38$ and zero. Also, \pm infinity and not-a-number (NaN) are represented.

Representation

Single precision floating point numbers are represented in four bytes. The format is as follows:

Most significant	Byte 2	Byte 3	Least significant
S E E E E E E E	E M M M M M M M	M M M M M M M M	M M M M M M M M

S = sign bit, 0 = positive, 1 = negative

E = binary exponent with bias of 128

M = mantissa with 23 explicit bits and an implied 1 bit as the most significant bit

In the Tattletales, the most significant byte is at the lowest address of the four bytes. This is important if binary data is transferred directly to a computer that interprets numbers in the opposite order (as all computers with Intel microprocessors do).

Variables

When using a floating point variable for the first time, append an exclamation point (!) after the last character of the name. The exclamation point does not become part of the variable's name and does not need to be included in later references to this variable. It simply allows the tokenizer to check that the correct types are being used for arguments. It also allows the tokenizer to automatically convert data to the correct representation.

Example: `sineValue! = sin(90.0)`
 `sineSquared! = sineValue * sineValue`
 `dim sineList!(100)`

Notice that sineValue and sineSquared have the ! suffix only the first time it is used. You may want to initialize all of your floating point variables near the beginning of the program and use that opportunity to use the ! suffix to declare them as floats. Finally, notice how sineList was declared as a floating point array. This MUST be done in the DIM statement because this is the first place an array is used.

NOTE: If an integer value is assigned to a floating point variable, the value will automatically be converted to float before the assignment.

Constants

Floating point numbers can be entered by following this rule. The entered number must contain at least one digit with: 1) a decimal point and/or 2) the power-of-ten operator E followed by the power of ten value.

Examples: 250., 2.5E2, 2500.0E-1 and 25E1 are all valid representations of 250.0
250, 2.5E2.0, E5 and .E5 are all INVALID floating point constants

NOTE: If a floating point value is assigned to an integer variable, the floating point value will first be converted to integer (using the FIX function) and then assigned to the variable.

Functions

Table 4-8 the functions available for floating point operations. The functions that take a floating point argument will automatically convert an integer to float before executing the function. Functions that take an integer argument will signal an error if passed a floating point argument (except for ABS which can take either type of argument - and returns the same type).

Table 4-8: Floating Point Operation Functions

Name	Description	Argument type	Return type
abs	absolute value	float or int	float or int
aint	closest integer less than argument	float	float
asflt	intepret data as float (no conversion)	integer	float
atn	arctangent (in degrees)	float	float
cos	cosine of angle in degrees	float	float
exp	raise e to power	float	float
fix	float to integer closer to zero	float	integer
float	integer to float	integer	float

Table 4-8: Floating Point Operation Functions (Continued)

Name	Description	Argument type	Return type
int	float to integer less than argument	float	integer
log	natural logarithm	float	float
log10	common logarithm	float	float
sin	sine of angle in degrees	float	float
sqr	square root	float	float
tan	tangent of angle in degrees	float	float

All other TxBASIC functions (CHAN, COUNT, PERIOD, PIN, TEMP, VGET) take integer arguments and return integer values. The functions LABPTR, PEEK and VARPTR are also integer functions but are not available on the Model 8.

Arithmetic

All but one of the normal TxBASIC arithmetic operators are available for floating point math. The exception is the modulo operator '%'. The tokenizer checks the two operands of each arithmetic operation. If one is a float and the other an integer, the integer is first converted to float and then the operation is performed. The result is considered as type float in the rest of the expression. For example, in this equation:

$$\text{floatValue!} = \underbrace{5 * 2}_{\text{Integer}} + \underbrace{3 * 7.9}_{\text{Float}}$$

Float

5 and 2 are multiplied as integers (and saved temporarily). Then 3 is converted to float and then multiplied (as floating point) by 7.9. Then the result of 5 * 2 is converted to float and this is added to the (already float) result of 3 * 7.9 and stored as a float in floatValue. If instead we had assigned the result to an integer variable, the only change would be that the final result would be converted to integer just before the assignment; the math operations would not have changed.

Relational operations

All TxBASIC relational operators are available for floating point comparisons. The precedence is exactly the same, too. As with the arithmetic operators, the tokenizer checks the two operands of each compare operation. If one is a float and the other an integer, the integer is first converted to float and then the comparison is performed. Unlike math operations, relational operations do not carry the result (or type) of comparisons along in complex expressions. The result of each comparison can only be TRUE or FALSE.

For example, in this equation:

Float Integer
if $\overbrace{2.3 > 2}^{\text{Float}} \ \& \ \overbrace{18 < 20}^{\text{Integer}}$ print "TRUE"
Bit-wise AND

2 is converted to a float and then compared with 2.3 and the result, TRUE, is saved. Then 18 is compared with 20 (as integers) and this result is TRUE also. Finally, the two TRUE results are ANDed together (the & operator). This is TRUE and the 'print' will be executed.

NOTE: Beware of checking for equality in an IF statement using floating point operands. It's possible that the numbers may appear to be equal (using print commands) but may not truly be equal. You should always test using greater than or equal or less than or equal.

Conversions

There are two kinds of conversions: implicit and explicit. We have mentioned implicit conversions in the previous discussions. If two operands are of different types and one is converted to the other's type, this is an implicit conversion. TxBASiC only makes implicit conversions from integer to float. Explicit conversions are accomplished with the FLOAT, INT and FIX functions. You must watch out for overflow and loss of precision when making explicit conversions. Overflow occurs when a floating point number with a large exponent is converted to integer. For instance, 1.0E15 cannot be converted to integer because the maximum integer is approximately 2.1E9. This is considered an error by TxBASiC. Loss of precision occurs when an integer with too many digits is converted to floating point. Integers in TxBASiC have 31 bits of precision (plus a sign bit) while floating point numbers only have a precision of 24 bits (one bit is an implied 1). TxBASiC does not consider this an error. It just rounds the floating point value to the nearest representation of the integer that it can.

There is an ambiguous situation. This is when floating point values are stored in binary in the datafile or in EEPROM. Anything stored there is assumed to be integer. In this case you must use the ASFLT function to tell TxBASiC to interpret the data as floating point. See ASFLT on [page 5-14](#).

Print/Store formats

There are two floating point formats for printing or storing data as characters. Fixed point notation uses only digits and the decimal point. Scientific notation uses a mantissa (which is really fixed point) followed by an exponent (in powers of ten). The format statement in PRINT and STORE commands for fixed point is:

#w.dF

Where **w** is the minimum field width of the number (including the sign, integer part, decimal point and fractional part) and **d** is the number of decimal places in the fractional part of the number. For instance, the number -15.302 would be specified as #7.3F.

The format for scientific notation is very similar to fixed point:

#w.dS

But here, the minimum field width, **w**, must also include the exponent specifier. For instance, 2.1345E-3 would be specified as #9.4S. The **w** specifier is optional (it defaults to a minimum field width of zero) and the **d** specifier is optional (it defaults to six decimal places) but if **d** is specified, it **MUST** be preceded by the decimal point. Here are some examples of valid formats and what they produce:

Format	Number = 12.345	Number = -0.987654
#10.2F	12.34	-0.99
#7.2S	1.23E1	-9.88E-1
#6.3F	12.345	-0.988
#.5F	12.34500	-0.98765
#F	12.345000	-0.987654
#S	1.234500E1	-9.876540E-1

Floating Point Errors

Errors in floating point operations do not stop program execution. Instead, a bit is set in the read-only variable FPERR. These bits remain set until you access FPERR by copying it to a variable, printing it or storing it. Then FPERR is cleared to zero. If you perform a number of floating point operations that produce different errors, FPERR will have more than one bit set when you finally check its value. The drawing to the right shows which bit is set for each of the four possible floating point errors. When the value of FPERR is printed, each bit takes on the following values: bit 0 = 1, bit 1 = 2, bit 2 = 4 and bit 3 = 8. The values are then added up to produce the final value (just as in any binary number). For instance, if Underflow, Not-a-Number and Loss of Precision errors occur before FPERR is printed, the value 13 (1 + 4 + 8) will be printed.

Here is the explanation for each error:

Underflow - Bit 0 (weight = 1) is set if a number between +1.175494E-38 and zero results or a number between -1.175494E-38 and zero results. This is not representable in single precision. The result is set to zero.

Example:	floatValue! = 3.0E-28 * 3.1E-15 print "result = ", #1F, floatValue print "error = ", FPERR	<i>produce the error print the result print value of FPERR</i>
Printed:	result = 0.000000 error = 1	<i>value of floatValue value of FPERR</i>
Overflow -	Bit 1 (weight = 2) is set if a number greater than +3.402823E+38 or less than -3.402823E+38 resulted. This is not representable in single precision. The result is set to +Infinity or -Infinity.	
Example:	floatValue! = -2.0E30 * 1.0E20 print "result = ", #1F, floatValue print "error = ", FPERR	<i>produce the error print the result print value of FPERR</i>
Printed:	result = -INF error = 2	<i>value of floatValue value of FPERR</i>
Not-a-Number -	Bit 2 (weight = 4) is set when the floating point routine has no idea of how to represent the result. One way to get this is to take the square root of a negative number which is an imaginary number. The result is flagged as a 'not-a-number'.	
Example:	floatValue! = sqr(-4.0) print "result = ", #1F, floatValue print "error = ", FPERR	<i>produce the error print the result print value of FPERR</i>
Printed:	result = NaN error = 4	<i>value of floatValue value of FPERR</i>
Loss of Precision -	Bit 3 (weight = 8) is set when an integer greater than 16777215 or less than -16777215 is converted to floating point. This is because it takes more than 24 bits to represent such a number and single precision floating point has only 24 bits to represent the precision of a number. TxBASIC will convert the value to the nearest floating point equivalent.	
Example:	floatValue! = float(17789321) print "result = ", #1F, floatValue print "error = ", FPERR	<i>produce the error print the result print value of FPERR</i>
Printed:	result = 17789319.000000 error = 8	<i>value of floatValue value of FPERR</i>

Converting TT BASIC Programs to TxBASIC

For some people, the only reason for using TxBASIC is to get an old TT BASIC program to run faster or to take up less space. Most people don't want to take the time to rewrite their program from scratch. The TT BASIC program usually works with some modifications.

One of the design goals for TxBASIC was to give it the ability to execute as much TTBASIC code as possible. There will be some TTBASIC programs that just will not run in TxBASIC without major modifications but you will find that most TTBASIC code will transfer to TxBASIC quickly. This will help you over some of the rough spots of converting to TxBASIC. We don't guarantee that this list is complete. A copy of this on the Onset Computer bulletin board will be updated for two reasons:

1. When additional problems, and solutions to the problems, are found and
2. When a problem is corrected in TxBASIC.

We would appreciate hearing from you if you find a problem and/or a solution when converting to TxBASIC.

Command names must be followed by a space or tab character	
TT -	You can use this form with no problem: 10 ITEXTX
Tx -	This would be a syntax error in TxBASIC. You would have to separate ITEXT from the X with a space or tab character.
Using labels and lengthy variable names	
The use of labels (instead of line numbers) and variable names up to 32 characters long (instead of simple A - Z variable names) is just a convenience and not a necessity. You do not need to remove the line numbers from your TTBASIC program to transfer it to TxBASIC; however, you will have to modify each line number and all references to those line numbers to start with a character.	
Loading a TxBASIC program removes any previous program	
TT -	It is possible to load selected portions of a program without overwriting the entire program. For instance, if you find that only line 70 is incorrect, you can send a new line 70 to the Tattletale without disturbing the rest of the existing program.
Tx -	This is not possible in TxBASIC. Any program load overwrites any existing code in the Tattletale's program area.
All Model 4A and 5 programs must have DFSIZE command on first line	
The tokenizer must know that a program is intended for a Model 5 because the memory map for the Model 5 is different from all other Tattletale models. The DFSIZE command:	
<ol style="list-style-type: none"> 1. Signals the tokenizer that a program is intended for a Model 5. 2. Reserves some of the Model 5's on-board RAM for use as datafile. 	

The NEW command is not available	
TT -	Some programmers put a NEW command at the beginning of a program they are sending to the Tattletale so that any previous program is cleared before the new program is loaded.
Tx -	You should just remove the NEW command from your program. All program loads overwrite any existing code in the Tattletale's program area. No immediate commands are allowed.
The RUN command has a different meaning	
TT -	Some programmers put a RUN command at the end of the TT BASIC program on their desktop computer so that when they send the program to the Tattletale it will start immediately.
Tx -	There is no way to accomplish this in TxBASIC. You must load the program into the Tattletale and then either select the “ Run ” command from the “ Tattletale ” menu or send a Ctrl-R character to the Tattletale to execute the program. No immediate commands are allowed.
Reason: The RUN command is used to start a background task in TxBASIC.	
The FRE, EON and EOFF are not available	
These immediate commands do not exist in TxBASIC. Do NOT attempt to embed them in your programs.	
The CON and LIST commands are not available	
TT -	A version of the LIST command (LIST ptrvar, x-y) allows storing the portion of the program from line x to line y in the datafile starting at location ptrvar. Then the CON {ptrvar} command transfers control to that portion of the datafile.
Tx -	This is not available in TxBASIC.
The ONLOAD command is not available	
One way to read data into the datafile in TxBASIC is with ITEXT. The trouble with this is that there is no error checking and ITEXT terminates on finding a specified character. Another option is to write a TxBASIC program that receives data from the UART and stores it in the datafile. You would then have to load the working program over this 'datafile loader' program.	
The BAT command is not available	
This command will NOT be added to TxBASIC. You can put a resistor divider between your battery terminals and use CHAN to read the center tap.	
NOTE: Make sure this value never goes above 5V. Also, the resistors should be large enough to not drain the battery but small enough to not slow the sample-and-hold acquisition of the A-D converter.	
The REM command has a slight difference	

Since TxBASIC variable names can be up to 32 characters long, we can't assume everything following the letters R, E and M is to be ignored. This could be part of a variable called 'REMEDY'. For this reason, TxBASIC forces you to put a space, tab or end-of-line (line feed or carriage return/line feed) character after REM.

Line numbers can NOT be used

Line numbers can NOT be used in TxBASIC. You can however have a one character label followed by a number to simulate a line number. For example: M1: could be used as the line number for line one of your program (TxBASIC would use it as a label).

GOTO and GOSUB accept only labels as an argument

TT -	<p>This program is possible:</p> <pre> 10 INPUT "Input channel" C 20 GOTO 1000 + C 1000 PRINT "LINE 1": GOTO 10 1001 PRINT "LINE 2": GOTO 10 1002 PRINT "Ending": STOP </pre> <p>You would probably want to check that C was 0, 1 or 2 after line 10.</p>
Tx -	<p>The TT BASIC example would have to be rewritten as follows:</p> <pre> start: INPUT "Input channel" C IF C = 0 PRINT "LINE 1": GOTO start IF C = 1 PRINT "LINE 2": GOTO start IF C = 2 PRINT "Ending": STOP </pre> <p>As with the TT BASIC code, you would want to have a default action if C is not equal to 0, 1 or 2.</p>
Reason: TxBASIC programs can not use line numbers.	
The PIN command returns a different value	
TT -	<p>The value returned depends on how many pins are listed in the command and the position the pins in the list. If I/O pins 1 and 2 are high and I/O pin 3 is low, the result of PIN (1, 2, 3) is not the same as the result of PIN (3, 2, 1).</p>

Tx -	TxBASIC does not know the order the pins were listed in because all the listed pins are combined into one number before PIN is executed. For this reason, each digital I/O pin is assigned a binary weight (refer to the PIN command on page 5-71 for additional information). The weight of each I/O pin in the PIN list is multiplied by its state (0 for low, 1 for high) and the results are added together and returned. For instance, the weights for pins 1, 2 and 3 are 64, 32 and 16 respectively. If pins 1 and 2 are high and pin 3 is low, PIN would return: $(64 * 1) + (32 * 1) + (16 * 0) = 96$. It is much easier if you just set up your program so that all tests of PIN are for zero or non-zero.
The DIM command is used differently	
TT -	There is only one array. The @ array. TT BASIC allows you to adjust the size of this array using the DIM command.
TX -	You can have as many arrays, of any size, as memory will allow. These arrays can be named any legal variable name. The DIM command is used by TxBASIC to define and set the size of new arrays (refer to the DIM command on page 5-31 for additional information).
Using the @ array	
TT -	The @ array defaults to 3071 elements for the Model 2B and 6. The @ array defaults to 1024 elements for the Model 4A and 5. These sizes can be modified using the DIM command.
Tx -	The @ array must be declared before it can be used. The DIM command is used to declare and set the size of the @ array - it has no default size. This is called 'dimensioning' the array. You are not limited to just the @ array in TxBASIC. You can have any number of arrays (up to the memory limit) using any legal variable name as long as you dimension each array first using DIM.
Using VSTORE and VGET is different	
TT -	Storing a value in one of the 32 EEPROM variables on a Model 2B or 6 takes two steps. First the value is stored in the appropriate @ array element. Then VSTORE is called with a reference to the @ array variable. VGET is similar. It stores the value in EEPROM in the corresponding @ array element. The value can then be read from the @ array.
Tx -	The @ array is not needed in the process of accessing EEPROM variables. VSTORE takes two arguments: the EEPROM address (0 to 31) and the value you want to store. VGET is used as a function that takes the EEPROM address as a parameter and returns the value at that address.

ASM code is handled differently	
TT -	Every line of assembly code must be preceded by the ASM command and include a variable that has been initialized with the address to which the code will be assembled. The only 'labels' that can be used for branching are TTBASIC variables that have been set to program addresses. If forward references are made, two passes must be made through the ASM code to resolve the 'label' values. There is no provision for in-line assembly code. TTBASIC variables cannot be accessed except as labels or through their absolute address.
Tx -	The ASM directive is only used once for a block of assembly code. The end of the block is marked with the 'END' directive. True labels (of up to 32 characters) are supported. The ASM mnemonics are fully assembled in the tokenizing process so that, when loaded into the Tattletale, it contains the 6301/6303 machine code. TxBASIC variables can be accessed by name.
In-line assembly code:	
These blocks start with the ASM \$ command and are ready to be executed as soon as the program is loaded. The program executes the TxBASIC tokens until it reaches the in-line assembly code where it executes 6301/6303 op codes. The program automatically switches back to token execution when it reaches the end of the assembly code. This code is not meant to be used as a subroutine and there is no way to initialize the processor's registers before the code begins to execute.	
Assembly code loaded to an address:	
This type of assembly code is more like TTBASIC assembly code. This code is intended to be used as a subroutine called from TxBASIC via the CALL command. These blocks start with the ASM <address> command where <address> is a Tattletale address where the code will be loaded BEFORE it is executed. To load the code to that address, just execute this section of the TxBASIC program (once only). Then the code can be accessed from TxBASIC. Refer to the CALL command on page 5-19 to see how the processor registers can be initialized before the subroutine begins execution.	
ASM statements cannot be used to initialize a region of memory	
TT -	Short ASM statements can be used to initialize regions of memory. For instance, to store the value 7 in memory location 16: ASM 16, DB 7
Tx -	The TxBASIC assembler automatically adds an RTS command at the end of any code it assembles. Therefore, a 39H would be stored at location 17 in the example above. You should use the POKE command. To accomplish the same thing as the TTBASIC example above, use: POKE 16, 7

The UPROG command is not available																			
<p>This command will not be added to TxBASiC. It is possible to reformat the hex listing into an ASM statement but it's a lot of work. For instance, this UPROG statement:</p>																			
<pre>UPROG "0474C0000102030400</pre>																			
<p>Could be reformatted as:</p>																			
<pre>asm &H74C0 data.b 01H, 02H, 03H, 04H end</pre>																			
<p>NOTE: The TxBASiC assembler automatically adds an RTS command at the end of any code it assembles. Therefore, a 39H would be stored at 74C4H. For short blocks of hex code, use the POKE command. For instance:</p>																			
<pre>POKE &H74C0,1: POKE &H74C1,2: POKE &H74C2,3: POKE &H74C3,4</pre>																			
Parameters are passed to ADLOOP differently																			
TT -	@ array elements 32 through 39 are used to pass parameters to the ADLOOP command.																		
Tx -	<p>A dedicated array called ADLOOP is used to pass parameters to the ADLOOP command. Do NOT dimension the adloop array. Like the ? array, the ADLOOP array is predefined by TxBASiC. This list shows the conversions:</p> <table border="0"> <thead> <tr> <th>TTBASiC uses</th> <th>Convert to this for TxBASiC</th> </tr> </thead> <tbody> <tr> <td>@(32)</td> <td>adloop(0)</td> </tr> <tr> <td>@(33)</td> <td>adloop(1)</td> </tr> <tr> <td>@(34)</td> <td>adloop(2)</td> </tr> <tr> <td>@(35)</td> <td>adloop(3)</td> </tr> <tr> <td>@(36)</td> <td>adloop(4)</td> </tr> <tr> <td>@(37)</td> <td>adloop(5)</td> </tr> <tr> <td>@(38)</td> <td>adloop(6)</td> </tr> <tr> <td>@(39)</td> <td>adloop(7)</td> </tr> </tbody> </table>	TTBASiC uses	Convert to this for TxBASiC	@(32)	adloop(0)	@(33)	adloop(1)	@(34)	adloop(2)	@(35)	adloop(3)	@(36)	adloop(4)	@(37)	adloop(5)	@(38)	adloop(6)	@(39)	adloop(7)
TTBASiC uses	Convert to this for TxBASiC																		
@(32)	adloop(0)																		
@(33)	adloop(1)																		
@(34)	adloop(2)																		
@(35)	adloop(3)																		
@(36)	adloop(4)																		
@(37)	adloop(5)																		
@(38)	adloop(6)																		
@(39)	adloop(7)																		
Variables are not at fixed locations																			
TT -	The A-Z variables and the @ array are all at fixed locations in memory because there is no choice about naming variables or defining arrays.																		
Tx -	You can name variables anything you like and they can be defined in any order. Also, you can have as many arrays as memory will hold. It's impossible to predefine a location for variables. Use the symbol table or the VARPTR function to find variable addresses.																		

The FOR-NEXT loop acts a little differently	
TT -	<p>The test for continuation and the calculation is done at the end of the loop. Thus, if you have this loop:</p> <pre style="text-align: center;">FOR J = 0 to -1 <BASIC code> NEXT J</pre> <p>The loop will be executed once. Also, the step size and the limit for the variable are calculated once at the beginning of the loop and never recalculated.</p>
Tx -	<p>The test for continuation is done at the beginning of the loop (as in most other languages) so the <BASIC code> in the example above would never be executed in TxBASIC. Also, the step size and limit for the variable are calculated each time the loop is executed. This slows the loop slightly but allows you to exit it with a GOTO no matter how deeply nested.</p>
General User Areas are in a different locations.	
TT -	<p>User RAM area #1 (mostly used for assembly code) starts at 112H and User RAM area #2 starts at 4000H on Model 2B and 6 and at 72BCH on the Model 5.</p>
Tx -	<p>General User Area #1 starts at 118H because TxBASIC includes two extra interrupt jump vectors. These are the Trap interrupt at 112H (which can be used to implement a trace routine in assembler) and the Software interrupt at 115H (which you can use for a custom action when the SWI opcode is executed. General User Area #2 starts at 74C0H (just above the Variables Area.</p>
INPUT statements expecting integer values accept only numerical values.	
TT -	<p>This is allowable:</p> <pre>A = 100 INPUT B PRINT B</pre> <p>When the program prompts with 'B:', you can type 'A' and the PRINT will produce 100.</p>
Tx -	<p>No record is kept of the name of a variable once your program is loaded. All TxBASIC knows of a variable is its address in memory. TxBASIC has no idea what 'A' refers to.</p>
The Assembler accesses data from BASIC variables differently	
TT -	<p>You must precede the variable name with the # symbol.</p>
Tx -	<p>Just use the TxBASIC variable name as you would an assembly variable name. Just remember - TxBASIC variables are four bytes long and the name refers to the most significant byte.</p>

TxBASiC Power-up Program Launch

When you first powered up your Tattletale, it printed a sign-on message similar to:

```
Tattletale Model X.XX  
TxBASiC Version X.XX  
(C) 19XX Onset Computer Corp  
  
#
```

The Tattletale is running a program written in TxBASiC that was stored in EPROM (in EEPROM on the Model 5F, 5F-LCD, 6F and 8). This program ends with a STOP command so that the prompt will appear. This section shows you how to make your program launch on power-up instead of the one that prints the message shown above. When we speak of ROM in this section, we are referring to EEPROM on the Model 5F, 5F-LCD, 6F and 8 and EPROM in all other Tattletaes.

NOTE: TxBASiC programs, no matter how launched, will break with a CTRL-C unless a CBREAK command has been used to specify a line number to restart to when a CTRL-C is encountered. You can disable CTRL-C breaks by writing a zero byte to address 9E hex. A count of CTRL-C characters will continue to be updated at address 93 hex. Clear this before you re-enable break-outs. Refer to the CBREAK command on [page 5-21](#) for additional information.

The power-up sequence for TxBASiC it is the same for all models.

On power-up TxBASiC calculates the checksum of the program in RAM and compares it with a stored value. If it matches it launches that program. If no match is found it boots the program in ROM into RAM and launches that program. Remember that the program will always break out upon finding a CTRL-C unless a CBREAK command has been used to specify a label to vector to when a CTRL-C is encountered.

TxBASiC has two program launch commands. The Run command in TxTools (or a CTRL-R sent directly to the Tattletale) assumes that the RAM-resident program is correct, calculates the correct checksum and installs it so that it will power up from RAM unless the program area has been modified or corrupted.

NOTE: Any change to this program RAM area after launch will make the checksum invalid.

The Boot EPROM command in TxTools (or a CTRL-B sent directly to the Tattletale) forces an incorrect checksum into RAM so that the program will thereafter always boot from ROM on power-up.

Loading your own Programs into the EPROM (Model 2A, 2B, 5 and 6 only)

The EPROM of the Model 2A, 2B, 5 and 6 contains the operating system as well as the user's program. When you erase the current user's program that prints the sign on message you will erase the operating system as well. The following procedure explains how to make a copy of the operating system and the user program at once.

NOTE: If you are using a Model 2A, 2B or 6 we strongly recommend that you buy another EPROM (27C256) so you can keep your original. This will let you recover from a disaster without having to get another copy of the operating system.

1. Load the Tattletale with your fully-debugged program (Use the Load command from TxTools).
2. While you are at the # prompt on the Terminal Window, pull down the “**Tattletale**” menu and select “**Remind EPROM**”.

NOTE: The Remind command does not work on Macintosh computers.

3. The “Save As” dialog box will appear. Enter a name and click the OK button or press ENTER. TxTools will off-load a hex file that contains the TxBASiC operating system (stored between C000H and FFFFH), and the program suitably offset so that it can be burned into the 27C256 EPROM.

On power-up, only the part of the EPROM containing the program is copied to RAM. This section of EPROM starts at 0140H (8140H on the Model 5).

4. Burn this Hex file into your erased EPROM by using our ReMinder™ programmer with the "[Using The ReMinder™ Programmer \(Model 2A, 2B, 5 and 6 only\)](#)" procedure.

Using The ReMinder™ Programmer (Model 2A, 2B, 5 and 6 only)

A program developed and tested in RAM can be burned into an erased EPROM so that it will be loaded into RAM and launched on power-up. When programming the Model 5, you must have the B-5 adapter (sold separately).



NOTICE



There are some programmers that can damage the Tattletale or the EPROM. We recommend using our ReMinder™ programmer to avoid damage.

When using somebody else's programmer for the Model 5:

For a Model 5 you must erase and burn the whole Model 5 since the EPROM is not removable. The B-5 adapter makes the Tattletale look like a 27C256. Some programmers apply 12 volts to address line A9 to cause the EPROM to read its manufacturer and part numbers. On the Model 5 address line A9 is connected to not only the EPROM but the RAM and 6303 as well. The 12 volt level could easily damage both the RAM and 6303! Most of these programmers have a method of suppressing the 12 volt signal, but others can destroy a Model 5.

Refer to the ReMinder operation manual for further instructions.

Loading your own Programs into the EEPROM (Model 5F, 5F-LCD and 6F only)

A program developed and tested in RAM can be burned into the Flash EEPROM so that it will be loaded into RAM and launched on power-up.

NOTE: Erasing the Flash EEPROM also erases the operating system. Make sure that you have a copy of the operating system before you erase the Flash EEPROM.

Offloading Your Program and the Operating System from RAM

NOTE: This procedure is applicable to all Tattletale models except the Model 8. The Model 8 uses the Remind EEPROM command to burn programs permanently into the EEPROM that are currently loaded in RAM.

This procedure offloads both the program and the operating system from the Tattletale to your hard drive. Once you have developed, debugged and loaded your program, perform the following:

NOTE: You can't be running your program while performing this procedure.

1. Start the TxTools program.
2. While you are at the # prompt on the Terminal Window, pull down the "**Tattletale**" menu and select "**Remind EPROM**".

NOTE: The Remind command does not work on Macintosh computers.

The "**Receive Program + TxBASIC**" window will open. The cursor will be in the name dialog box.

3. Type a name for your program. Record the location of where the file will be saved.
4. Click the OK button. The file will be offloaded and saved on your hard drive. A status box will display the saving progress along with a bar graph. If it displays "Too Many Errors Receiving Packets", perform the following:
 - a. Turn off the power to the Tattletale for a few seconds.
 - b. Turn on the power to the Tattletale.
 - c. Repeat steps 2 - 4 and try again.

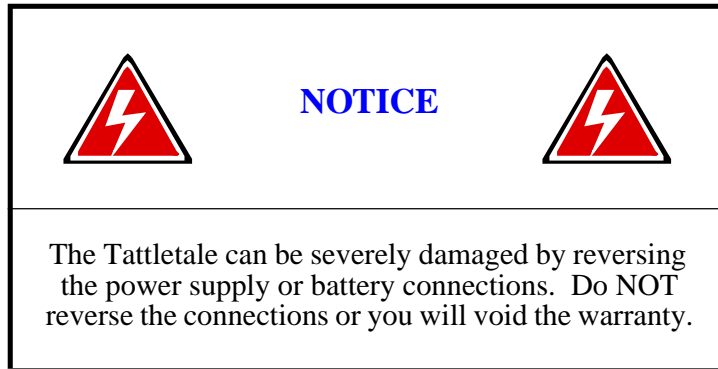
Once the Remind EPROM option is successful, your program and the TxBASIC operating system are saved in the hex file together. The hex file that is produced is exactly what is needed to be burned back into the EEPROM so that your program will be launched on power-up along with TxBASIC.

5. Perform the "**Erasing the Tattletale EEPROM (Model 5F, 5F-LCD and 6F only)**" procedure.

Erasing the Tattletale EEPROM (Model 5F, 5F-LCD and 6F only)

NOTE: A 12V \pm 0.6V power supply is required for this procedure.

1. Connect a 12V power supply to the Tattletale's power supply connection.



2. With TxTools already running, turn on the 12V power supply. The Tattletale should display the startup screen showing that it is in proper working order before erasing the EEPROM.
3. Turn off the 12V power supply.
4. Insert the programming shunt (included in the development kit) into the jumper connection on the Tattletale as shown in Figure 4-14, Figure 4-15 and Figure 4-16.

NOTE: If you are using the PR-5F board, it may be necessary to clip the corner of the programming shunt so that it will fit into the socket without hitting the communication cable connector on the PR-5F board. Using a bare wire to short the two sockets of the programming plug on the Model 5F, 5F-LCD or 6F will work fine. That's all the programming shunt does.

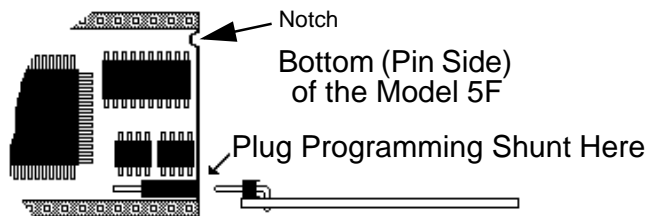


Figure 4-14: Where to Insert the Programming Shunt (Model 5F)

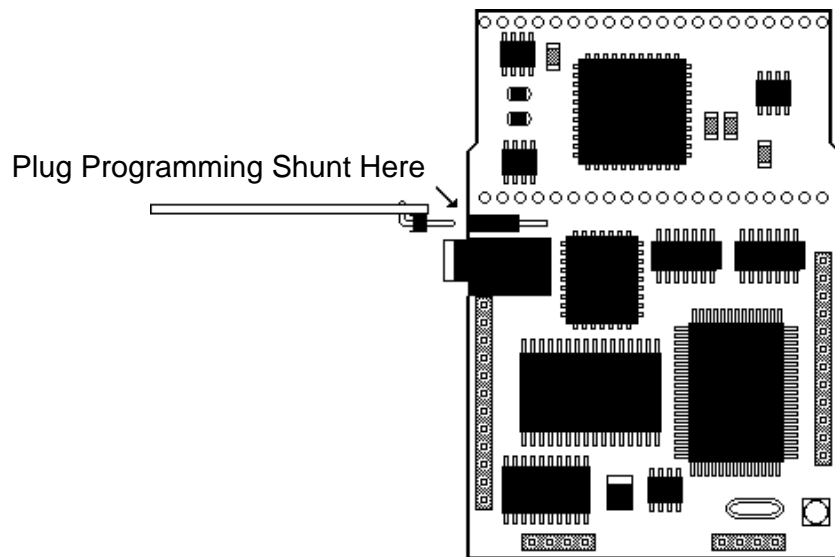


Figure 4-15: Where to Insert the Programming Shunt (Model 5F-LCD)

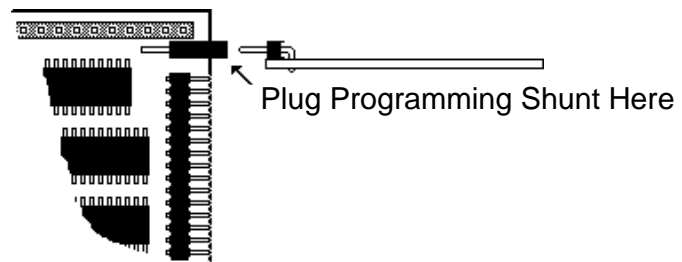


Figure 4-16: Where to Insert the Programming Shunt (Model 6F)

5. Turn on the 12V power supply. The following should be displayed:

```
MINIMONITOR (C) 1991 Onset Computer Corp., All Rights Resv.
!
```

With the CAPS LOCK key off and the cursor at the ! prompt, press SHIFT X (do not press Alt-X or the program will enter into hexadecimal mode). The screen should display the following:

```
Erasing... Erased: Max Burn = 01 Erasures = XXXX
```

The Tattletale will show the results of the erasing attempt. If it failed it will display the message "EPROM FAILURE". If you see that message, turn off the power to the Tattletale and then turn the power back on again, and try again. If you still get the error message, perform the **“Possible Problems Erasing an EEPROM (Models 5F, 5F-LCD and 6F only)” procedure on page 4-65**. Once the Flash EEPROM has been erased, you are ready to load your program into it.

6. Leave the programming shunt plugged in and perform the **“Loading your Program into the Erased EEPROM (Model 5F, 5F-LCD and 6F)” procedure**.

Loading your Program into the Erased EEPROM (Model 5F, 5F-LCD and 6F)

NOTE: The 12V power supply should still be connected to the tattletale and the programming shunt should still be inserted into the Tattletale.

1. With TxTools still running, pull down the “**CommPort**” menu (IBM PC) or the “**Terminal**” menu (Macintosh) and select “**SND File ASCII**”. An open file dialog box will appear.

NOTE: On the Macintosh, make sure that the “Send Prefix” and “Send Suffix” under the “**File Transfer Settings**” option are blank.

2. Select the file name you created on [page 4-61](#) (when you offloaded your program and TxBASIC) and click the OK button.

The file will be sent to the Tattletale. As it is sent, a dialog box will show the percentage complete as well as a bar graph.

3. Once the file is completely sent to the Tattletale, turn off the 12V power supply.
4. Remove the programming shunt.
5. Re-connect the battery or power supply (a 12V supply is no longer needed).
6. Once the battery or power supply is re-connected, the following should be displayed:

```
Tattletale Model X.XX
TxBASIC X.XX
(C) 19XX Onset Computer Corp., all rights reserved
#
```

The screen prompt should be a # symbol. If you press the RETURN or ENTER key it should display another # symbol.

Your program and the original operating system are now the main program in the EEPROM. Whenever the Tattletale is restarted, your program will be loaded and run automatically.

Loading your own Programs into the EEPROM (Model 8 only)

Storing your program permanently in the Model 8's EEPROM is simple with the following procedure:

1. Load your fully-debugged program with the Alt-L command (IBM PC) or Command L (Macintosh) from TxTools.
2. While you are at the # prompt on the Terminal Window, pull down the “**Tattletale**” menu and select “**Remind EPROM**”.
3. When you get the “Program burn successful” message, turn the main power off and on and watch your program start up.

Possible Problems Erasing an EEPROM (Models 5F, 5F-LCD and 6F only)

1. The power supply must be $12V \pm 0.6V$. If this voltage specification is not met an EEPROM Failure message may be displayed while trying to erase the EEPROM.
2. Occasionally, the Tattletale will power up in a random state causing the EEPROM Failure message to appear. Turn off the power supply and disconnect the communications cable for 10 seconds. Then reconnect the cable and turn it back on and try erasing the EEPROM again.

Possible Problems Erasing an EPROM (Model 5 only)

NOTE: If you plan to use somebody else's EPROM programmer; the Tattletale produces a hex file with values of A15 that reflect the actual address of the code in the running machine. This places the user program between 140H and 3FFFH (8000H and BFFF for the Model 5) and the operating system between C000H and FFFFH. Some EPROM programmers are finicky and get confused when they see A15 high since they know that the EPROM has no A15 input. If your programmer is one of those picky ones, you will have to load that section of the file (8000H and above) with an 8000H offset to keep your EPROM programmer from complaining.

The Onset Computer bulletin board has a program that will perform the job mentioned here. The executable program will run on IBM-PC or compatible computers. The source code is also available if you want to modify it for other computers. The program is called FIX32K.EXE and the source is in file FIX32K.C in the C programming language. Both are contained in an ARC formatted file called FIX32K.ARC.

NOTE: The program is not available for the Macintosh; however, the source code is available if you want to modify it for the Macintosh. Please call Onset Computer Product Support for additional help.



NOTICE



When using somebody else's programmer for the Model 5:

For a Model 5 you must erase and burn the whole Model 5 since the EPROM is not removable. The B-5 adapter makes the Tattletale look like a 27C256 except that some programmers apply 12 volts to address line A9 to cause the EPROM to read its manufacturer and part numbers. On the Model 5 address line A9 is connected to not only the EPROM but the RAM and 6303 as well. The 12 volt level could easily damage both the RAM and 6303! Most of these programmers have a method of suppressing the 12 volt signal, but if yours doesn't, you will destroy your Model 5 if you don't do it right.

Installing or Updating the TxBASIC Operating System

Loading TxBASIC into the EEPROM of the Tattletale (Model 5F, 5F-LCD and 6F)

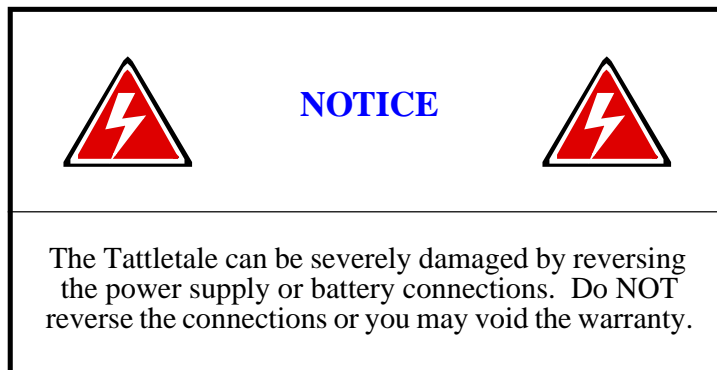
NOTE: This procedure only loads TxBASIC into the Tattletale, not your own BASIC program.

If your Tattletale Model 5F came programmed with TTBASIC instead of TxBASIC, you can easily install TxBASIC by using this procedure. Make sure you have a copy of the TxBASIC operating system before you erase the EEPROM.

NOTE: A 12V $\pm 0.6V$ power supply is required for this procedure.

Erasing the Tattletale EEPROM (Model 5F, 5F-LCD and 6F)

1. Connect a 12V power supply to the Tattletale's power supply connection.



2. Connect the communication cable to the Tattletale (the other end should still be connected to the computer).
3. Start the TxTools program.
4. Turn on the 12V power supply. The Tattletale should display the startup screen showing that it is in proper working order before erasing the EEPROM.
5. Turn off the 12V power supply.
6. Insert the programming shunt (included in the development kit) into the jumper connection on the Tattletale as shown in Figure 4-14, Figure 4-15 and Figure 4-16.
7. Turn on the 12V power supply. The following should be displayed:

MINIMONITOR (C) 1991 Onset Computer Corp., All Rights Resv.
!

With the CAPS LOCK key off and the cursor at the ! prompt, press SHIFT X (do not press Alt-X or the program will enter into hexadecimal mode). The screen should display the following:

Erasing... Erased: Max Burn = 01 Erasures = XXXX

The Tattletale will show the results of the erasing attempt.

If it failed it will display the message "EPROM FAILURE". If you see that message, turn off the power to the Tattletale and then turn the power back on again, and try again. If you still get the error message, perform the **“Possible Problems Erasing an EEPROM (Models 5F, 5F-LCD and 6F only)” procedure on page 4-65**. Once the Flash EEPROM has been erased, you are ready to load your program into it.

8. Perform the **“Loading TxBASIC into the Erased EEPROM (Model 5F, 5F-LCD and 6F)”** procedure.

Loading TxBASIC into the Erased EEPROM (Model 5F, 5F-LCD and 6F)

1. With TxTools still running and the programming shunt still inserted, pull down the **“CommPort”** menu (IBM PC) or the **“Terminal”** menu (Macintosh) and select **“SND File ASCII”**. An open file dialog box will appear.

NOTE: On the Macintosh, make sure that the **“Send Prefix”** and **“Send Suffix”** under the **“File Transfer Settings”** option are blank.

2. Select the appropriate file name for your Model (For example: TX5F-X.XX.HEX is for the Model 5F) and click the OK button. The X.XX should be the latest version of TxBASIC.

The TxBASIC hex file will be sent to the Tattletale. As it is sent, a dialog box will show the percentage complete as well as a bar graph.

3. Turn off the 12V power supply.
4. Remove the programming shunt.
5. Re-connect the battery or power supply (a 12V supply is no longer needed).
6. Once the battery or power supply is re-connected, the following should be displayed:

```
Tattletale Model X.XX
TxBASIC X.XX
(C) 19XX Onset Computer Corp., all rights reserved
#
```

The screen prompt should now be a # symbol. If you press the RETURN or ENTER key it should display another # symbol.

TxBASIC is now the main program in the EEPROM and the installation of TxBASIC is complete.

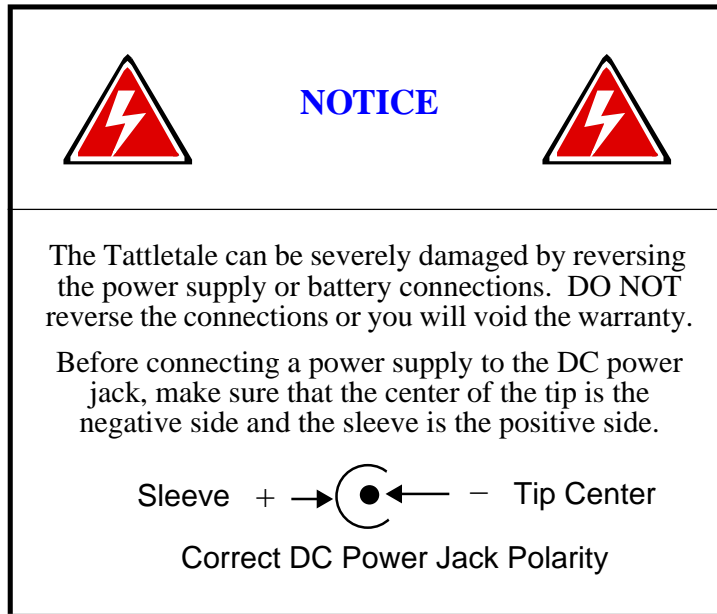
You can now use any of the commands in **Section 5 - TxBASIC Command Reference**.

Loading TxBASIC into the EEPROM of the Model 8

The TXBASIC directory contains the S-record file for the TxBASIC language.

This S-record is used to burn TxBASIC in the Model 8 EEPROM so that when you power-up, TxBASIC will run automatically. This file is called TXBASIC.AHX.

1. Start the TxTools program.



2. Connect a battery or a 7 to 15V power supply to the prototyping board's DC power jack. Make sure that the polarity is not reversed when you connect the power to the board, or you may severely damage the board.
3. If TxBASIC is not currently burned into the Model 8, you should see the "TOM8>" prompt. Pressing the RETURN key should show another TOM8> prompt. If the prompt is either TxB# or just the # prompt, then TxBASIC is already burned in. If you want to re-load TxBASIC just enter "**Cntl-X**" and then enter "**Y**" to the question displayed to exit to the TOM8> monitor.
4. From the TOM8> prompt, enter the command "**lo**" and press the RETURN key. A line of text reading: "Waiting for S-Records" will appear.
5. Go to the "**CommPort**" menu at the top of the screen (either by clicking on it with the mouse or using the Alt-C key combination) and select the item - labeled "**Snd file ASCII**". You will see a small file-finder box open. Find and open the file in the TxBASIC directory by typing "**c:\TXBASIC\TXBASIC.AHX**" in the "Name" box and press the RETURN key.

NOTE: If any errors are displayed after the file is loaded, you probably forgot to enter the "lo" command in step 4. Also, if you use the TXBASIC.RHX file by mistake, TxBASIC will only be loaded into RAM and will be lost as soon as the power is turned off.

The 'c' character should be replaced with the designation of the disk drive containing

the TxBASIC directory.

A status dialog will appear showing the progress of the load. TxBASIC is quite large and will take about 5 minutes to load. After the file is loaded, you will see a message like:

```
Target is Flash!
start addr = 00002000
end addr = 00016181
Ok to write flash between above addresses? (Y/N)
```

6. Enter a “Y” and you will see a string of asterisks and finally see the TOM8> monitor prompt again.

```
FLASH ID = ATMEL 29C010
burning*****
TOM8>
```

7. Now you can power-off the Tattletale and power it up again. The following TxBASIC sign-on message will be displayed ending with the TxBASIC prompt:

```
Tattletale Model 8, TxBASIC Version X.XX
(C) 19xx Onset Computer, Pocasset, MA, USA

TxB#
```

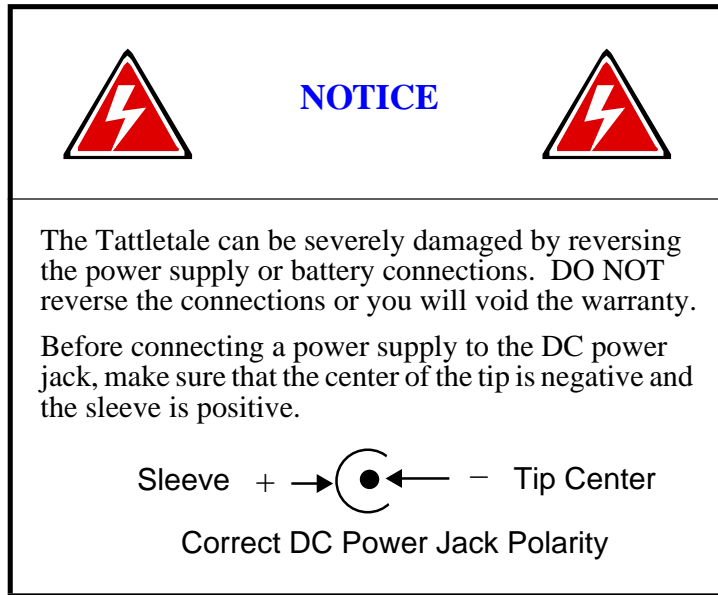
Press the RETURN key several times to make sure the TxB# prompt repeats which means everything is okay.

How to Completely Erase the Flash EEPROM Memory (Model 8 only)

NOTE: This procedure completely erases the flash EEPROM - including TxBASIC.

1. With the power to the Model 8 turned off, start the TxTools program.
2. If you have a program in Flash memory that disables Control-C and is still running, perform the following to force it to stop executing and return to the TOM8> prompt.
 - a. Before connecting power to the Model 8, connect a jumper from pin A5 (-IRQ3) to pin A1 (DGND). This will force the Model 8 to ignore any programs in the flash memory and go to the TOM8 monitor when the power is connected.

NOTE: If you are using the I/O-8 board you can separate the I/O-8 from the Tattletale slightly so that you can connect the jumper properly to the pins of the I/O-8 board.



- b. Connect the power source to the Model 8 (with the jumper still connected). The TOM8> prompt will be displayed. With -IRQ3 tied to GND, the LED on the Model 8 will turn on.
 - c. Proceed to step 4.
 3. If you have a program burned in flash memory (but not running) and want to erase all of the flash memory, perform the following:
 - a. From the Terminal window at the TxB# prompt press the key combination “**Cntl X**”.
 - b. The computer will display the question “Reset to monitor (Y/N)?”. Enter “**Y**” and the TOM8> prompt will be displayed.
 4. Enter the command “**lo**” and press the RETURN key. The computer will display “Waiting for S-Records”.
 5. Pull down the “**CommPort**” menu and select the “**Snd file ASCII**” option. An open file dialog box will open.
 6. Locate the file named “**CLRFLASH.RHX**” in the UTIL sub-directory of the TxTools directory and open the file. The file will be loaded onto the Model 8. After the load is complete it will display that the load was successful and return to the TOM8> prompt.
- NOTE:** If any errors are displayed after the file loads, you probably forgot to enter the “lo” command in step 4.

7. Enter the letter “g” and press the RETURN key. The current map of the Flash memory will be displayed and then the program will ask “Are you sure you want to erase the Flash EEPROM (Y/N)?”, enter “Y”. The program will completely clear the EEPROM, except for the TOM8 monitor, so anything you had loaded will be deleted (including TxBASIC if it was previously loaded). A status will be displayed as the EEPROM is cleared.

The TOM8> prompt will be displayed after the flash is cleared.

8. Turn off the power to the Model 8 and remove the jumper wire from pin A1 and A5.
9. If you separated the I/O-8 from the Tattletale slightly to attach the jumper, push the pins of the I/O-8 board all the way into the sockets of the Model 8.
10. Perform the "**Loading TxBASIC into the EEPROM of the Model 8**" procedure.

Section 4 - Model 8 TxBASIC Addendum

Introduction

This addendum is for Model 8 users of TxBASIC who need to create special functions not available with the standard TxBASIC.

NOTE: All TxBASIC commands are actually written in C using Aztec C as the compiler. If you do not have the Aztec C, you will have to purchase it from Onset Computer.

Build Your Own Basic (BYOB)

BYOB allows you to write functions in AZTEC C that are linked with the TxBASIC libraries and can easily be called from Onset's TxBASIC, adding extra functionality to your TxBASIC.

NOTE: Make backup copies of TxBASIC and all BYOB files before starting this procedure.

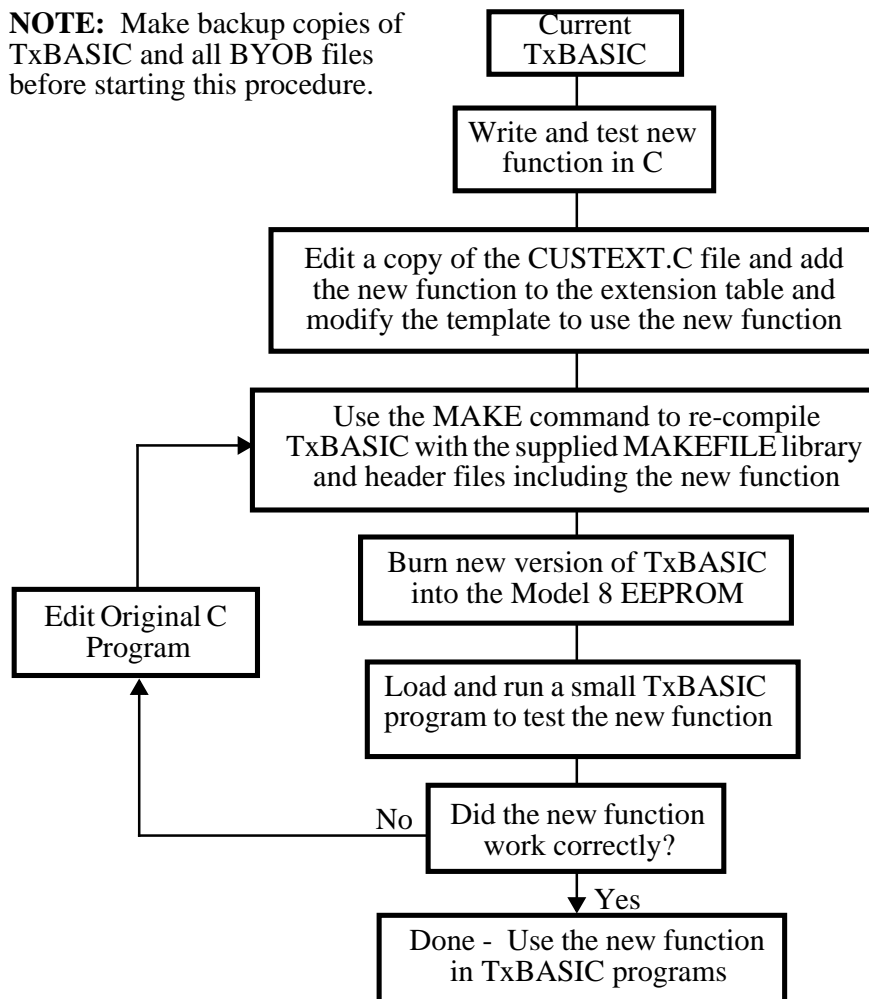


Figure 4-17: Flowchart for Creating your own TxBASIC Commands

These functions take a variable number of parameters specified by you. When invoked, a TxBASIC extension looks like a "C" function. For example, if you had written a random number generation function called **Random** - to invoke it you would use the following line :

```
<variable> = Random(<variable or constant>)
```

The parameters must be enclosed in parenthesis. The return value does not need to be assigned to a variable. Also the function names are **case sensitive** so be sure to match the case in all declarations and calls.

Creating the Function Random

The following code was adapted from Algorithms in C by Robert Sedgwick:

```
#define m 100000000
#define ml 10000
#define b 31415821

static long a = 5;

long Random ( long r )
{
    a = (mult(a,b)+1) % m;
    return ((a/ml)*r)/ml;
}

long mult(long p, long q)
{
    long p1,p0,q1,q0;

    p1 = p/ml;
    p0 = p%ml;
    q1 = q/ml;
    q0 = q%ml;

    return (((p0*q1+p1*q0)%ml)*ml+p0*q0) % m;
}
```

We have supplied the framework for adding functions to TxBASIC in the file **CUSTEXT.C**. You will need to modify this file and compile it using the MAKEFILE supplied. This will generate a new version of TxBASIC with the new extension in it. Before making any modifications to the file **CUSTEXT.C** be sure to back it up. When making modifications be sure only to modify the areas designated. Modifying any of the other code will most likely break the code.

This implementation of **Random()** will use either one or two parameters. If only one parameter N is specified the number returned will be in the range 0 to N-1. If two parameters are specified the second is a seed value. This is useful if you need to repeat the same random number sequence.

1. The first modification necessary to add an extension (function) to TxBASIC is to add the function's name to the extension table. Load the file **CUSTEXT.C** into your editor and look for the following lines of code under the heading **DEFINE EXTENSIONS HERE**:

```

*****
#ifdef BUILDING_EXTENSION_TABLES

    __ext__( xmplExtn )

#endif /* BUILDING_EXTENSION_TABLES */

```

2. Add the following function name entry below the existing entry:

```

    __ext__( Random )

```

The extensions table should now look as follows:

```

*****
#ifdef BUILDING_EXTENSION_TABLES

    __ext__( xmplExtn )
    __ext__( Random )

#endif /* BUILDING_EXTENSION_TABLES */

```

3. Next the actual code will be added. At the end of the file **CUSTEXT.C** is the following template:

```

#if 0 /* TEMPLATE */
/*****
** NEWEXT
** ===== YY/MM/DD ===
** expects:
*****
//.... global variables

//.... end of global variables
EXTENSION(NEWEXT )
{

//.... function prototypes
void test(long argc, long *argv);
//.... end of function prototypes
//.... local variables

//.... end of local variables
ExtExpectArgs(__min__, __max__);
//.... MAIN code

    test(argc,argv);
    return (__result__);

} /* NEWEXT() */
//.... end of MAIN code
//.... local functions
void test(long argc, long *argv){ }

// end of local functions and extension template
#endif /* TEMPLATE */

#endif /* ! BUILDING_EXTENSION_TABLES */

```

4. Select and copy the template text and paste it above the original.
5. Remove the upper and lower **#if** and **#endif** statements as well as the other text marked with a ~~strikethrough~~. Then add the code that is shown in **bold**. The remaining text is not changed.

```

#if 0 /* TEMPLATE */
/******
** NEWEXT EXTENSION RANDOM
**===== YY/MM/DD =====
** expects: one parameter to indicate integer range
and a second (optional) for a seed
*****
//.... global variables
#define m 100000000
#define m1 10000
#define b 31415821
static long a=5;

//.... end of global variables
EXTENSION( NEWEXT Random )
{
//.... function prototypes
void test(long argc, long *argv);
long mult(long p1, long p2);

//.... end of function prototypes
//.... local variables
//.... end of local variables
ExtExpectArgs( __min__, 1, __max__ 2 );
//.... MAIN code
test(argc,argv);
return(__result__);
// if two parameters second is seed
if (argc == 2)
a = argv[1];
a = (mult(a,b)+1) % m;
return (((a/m1)*argv[0])/m1);

} /* NEWEXT() Random() */
//.... end of MAIN code
//.... local functions
void test(long argc, long *argv){
long mult(long p, long q)
{
long p1,p0,q1,q0;
p1 = p/m1;
p0 = p%m1;
q1 = q/m1;
q0 = q%m1;
return (((p0*q1+p1*q0)%m1)*m1+p0*q0) % m);
}
// end of local functions and extension template
#endif /* TEMPLATE */

#endif /* ! BUILDING_EXTENSION_TABLES */

```

6. Exit the editor and type MAKE to build the new version of TxBASIC with the function **Random** included.
7. Load the file into the Tattletale 8's Flash EEPROM. To invoke it from TxBASIC try the following program:

```
//*****  
// TxBASIC EXAMPLE PROGRAM FOR MODEL 8  
//*****  
// DON'T FORGET TO SPECIFY MODEL 8!  
MODEL 800  
  
// DON'T FORGET TO SPECIFY EXTENSIONS USED!  
EXTENSION Random  
  
print "STARTING"  
  
print "Seeded starting value = ", Random(100,7)  
print  
for index = 1 to 100  
    number = Random(100)  
    print #4,number  
next index  
  
print "end"  
stop
```

After your new function is tested and proven to be working fine you can start using it in your TxBASIC programs.

NOTE: Onset Computer does not provide technical support for debugging the code of your new functions, only for problems with the hardware and software that we sell.

If you do need assistance in writing programs in C or TxBASIC or designing hardware, call the Onset BBS and download the information regarding our ICON (Independent Consultants Online Network) program. This program is made up of consultants all over the world who can be hired to assist you design hardware and software for the Tattletale. The files on the BBS explain the entire program.

NOTICE

Please note that Onset is in no way liable to either consultants or prospective customers with regards to pricing, performance, accuracy of published data, etc. Each project must be agreed upon by the two parties involved. We will continue to provide Tattletale hardware, software and normal technical support, but the final result is up to the consultant and client. We will perform only cursory screening of these forms; we will not necessarily vouch for the accuracy of sample code, circuits, etc.

Section 5 - TxBASIC Command Reference

How to Use this Section

This section of the manual is to be used for reference for all the TxBASIC commands. The commands are listed alphabetically. The structure of each command generally follows this format:

Command Name	Short description of command
Syntax of the command	
Description of the command	
Example of the command	
Remarks about the command	
Cautions about using the command	

Typical syntax lines in this section will look like these:

```
INPUT ["prompt"] var[;] [,["prompt"] var[;] ...]
INPUT ["<s1>"] <v1> [;] [,["<s2>"] <v2>[;]...]
```

At first glance these lines can be very confusing, but if the syntax lines are broken down into parts, the proper use of the command becomes clear. These syntax lines can be used as in Table 5-1. Anything in [] is considered optional so for the example shown the most basic use of the input command would be: Input A.

The examples in Table 5-1 are not all the possible uses of the command since you can use a “,” on the same line and have another input and so on. The syntax for multiple commands on one line uses the part that looks like: [,["**prompt**"] **var**[;] ...] - all of which is optional and is enclosed in [].

Table 5-1: Command Syntax Usage

Part of Syntax to Use	Example of Proper Command Use
INPUT var	Input A
INPUT "prompt" var	Input "prompt"A
INPUT var;	Input A;
INPUT "prompt" var;	Input "prompt"A;
INPUT "prompt" var; , "prompt" var;	Input "prompt "A; , "prompt #2 "B;
INPUT var, var	Input A, B

Table 5-2 shows the key for figuring out what the letters in the syntax examples mean. If we continue with the second line of the example:

```
INPUT ["<s1>"] <v1> [;] [,"<s2>"] <v2>[;]...
```

We can determine that this command requires only the command INPUT and a VARIABLE on the line to be used. If we decide to use the optional string it will be enclosed in " ".

Table 5-2: Syntax Usage Key

Letter in Syntax Example	Description of What the Letter Means
S	String - Can be any character or word
X	Expression - Usually a formula or a calculation
V	Variable - Can be a single letter or whole words
STR\$	String variable or string constant
C	Constant - Must be a number
\$	Inline assembly code
Label	Used in place of line number references
[]	Data between the [] is optional with the command syntax

TxBASIC Command Quick Reference

Table 5-3: TxBASIC Command Quick Reference

Command	Description
Program Commands:	
ADLOOP	Start background A-D routine to datafile buffer (not the Model 8)
ASM \$...end ASM<addr>...end	Assemble 6303 code inline (not the Model 8) Assemble 6303 code at <addr> (not the Model 8)
CALL x1,x2[,v] CALL x1,"s" [#n] [,x] [,x] [, {x,y}] [;]	Call x1 with registers = x2; returning in v (not the Model 8) Repeatedly call x1 with characters (not the Model 8)
CBREAK label CBREAK	Go to label if Ctrl-C character detected Return Ctrl-C handling to default behavior
COUNT v COUNT	Start counting I/O line 13 transitions, store in v Stop background count (count can also be used as a function) (The Model 8 uses I/O line 4 instead of I/O line 13)
DFSIZE x	Set on-board datafile size to x (Model 4A, 5 and 8 only)
DIM <label> (size)	Dimension variable <label> to 'size'

Table 5-3: TxBASIC Command Quick Reference (Continued)

Command	Description
DISPLY "s"[\,<x>] [,#n][,val] [,{\<x2>, <x3>}] [;]	Display 4-character string on LCD (5F-LCD only)
DOFFLD <x1>, <x2>, <v>	Offload data from disk (6, 6-1m and 6F only)
DSKON <v>	Power up disk (6, 6-1m and 6F only)
DSKOFF	Power down disk (6, 6-1m and 6F only)
DTOA <x>	Set dutycycle output (I/O line 12) (The Model 8 uses I/O line 6)
FOR v=x1 to x2 [STEP x3] NEXT v	Initiate iterative loop Re-enter "for" loop
GOSUB label RETURN	Execute subroutine at label Return from subroutine
GOTO label	Go to label
IF x command	Execute command if x = 1 (true)
IFF x ... [ELSE...] ENDIF	Execute 1st command block if x is true,else 2nd block
INPUT [s] v [;] [, [s] v [;] ...]	Prompt with s, load variable v
LET v1=x1 [,v2=x2 . . .]	Assign value to variable
MODEL <x>	Set Tattletale model type
ONERR label [,v] ONERR	Go to label if error encountered [, error in v] Return error handling to default behavior
POKE addr, value	Store byte 'value' at address 'addr'
POKEL addr,value	Copy four bytes of <i>value</i> to four bytes at address <i>addr</i> .
POKEW addr,value	Copy two bytes of <i>value</i> to two bytes at address <i>addr</i> .
PRINT "s" [,#n][,x][,\x][,{x,y}][;]	Send characters out main UART
QUIT	Enter "End of Program" low power mode (Model 6F only)
RATE x	Reduce SLEEP interval to 10mS / x
REM	Ignore rest of line
REPEAT ... UNTIL x	Repeat command block until x is true
RUN label	Start background routine at label (not the Model 8)
SCROLL "s"[\,<x>] [,#n][,val] [,{\<x2>, <x3>}] [;]	Scrolling LCD output (5F-LCD only)
STOP	End program execution
VSTORE x1,x2	Store x2 to EEPROM addr x1 (not the Model 4A or 5)
WHILE x ... WEND	Execute command block while x is true
XMIT- / XMIT+	Suppress / enable hardware UART output (not the Model 8)
XSHAKE x	Enable XON/XOFF with time-out x*10mS (not the Model 8)

Table 5-3: TxBASIC Command Quick Reference (Continued)

Command	Description
Datafile Storage Commands:	
BURST v,x,f	Store first x channels of A-D using pointer v , form f
DFSAVE <x1>,<v>[,<x2>]	Store datafile to disk (6, 6-1m and 6F only)
ITEXT v [,x1] [,x2]	Store UART's string to datafile using pointer v terminator x1, time-out x2; x2=0: 1 chr only
OTEXT v[,x] [;]	Send out UART string from datafile by pointer v with terminator x
STORE v, [#n],x...	Store x..., using n bytes, using pointer v
STORE v,"s",[#n][,x][,x][,{x,y}][;]	Store characters starting at location pointed to by v
UGET x1,x2,v,x3	Store x2 bytes of x1 baud input to I/O line 7 using pointer v; time-out x3 * 10mS (The Model 8 uses I/O line 14 instead of I/O line 7)
Datafile Retrieval Commands:	
DFREAD <x1>,<v>[,<x2>]	Read datafile from disk (6, 6-1m and 6F only)
GET (v, [#n])	Return integer formed from n bytes pointed to by v
GETS(v)	Returns a string from the datafile starting at the datafile address in variable var. This assumes the string was stored in binary form.
OFFLD x1,x2 [,x3,v]	Datafile off-load x1 – x2, time-out x3,errors in variable v
USEND x1,x2,v	Send x2 bytes at x1 baud out I/O line 8; pointer v (The Model 8 uses I/O line 13 instead of I/O line 8)
Functions:	
ABS(x)	Absolute value of x
AINT(x)	Integer part of x as float
ASFLT(x)	Interpret x as a floating point value
ATN(x)	Arctan of x
CHAN(x)	A-D conversion of channel x
COUNT(x)	Return # cycles at I/O line 13 in x*10mS period (It's also a command) (The Model 8 uses I/O line 4 instead of I/O line 13)
COS(x)	Cosine of x
EXP(x)	Return e raised to the x power
FIX(x)	Integer part of x as integer closer to zero
FLOAT(x)	Convert integer x to float
INSTR([x,] str1\$, str2\$)	Return the position in str1\$ at which str2\$ is first found. Optionally start the search at position x in str1\$.
INT(x)	Integer part of x as integer more negative
LABPTR (label)	Address of label (not the Model 8)
LEFT(str\$, x)	Return a string made up of the left-most x characters of str\$.

Table 5-3: TxBASIC Command Quick Reference (Continued)

Command	Description
LEN(str\$)	Return the length of str\$. This can be a variable or constant.
LOG(x)	Natural logarithm of x
LOG10(x)	Common logarithm of x
MID(str\$, x1, x2)	Return a string made up of x2 characters starting at character x1 in str\$.
PEEK(addr)	Byte at address 'addr'
PEEKL(addr)	Return four bytes at address addr .
PEEKW(addr)	Return two bytes at address addr .
PERIOD (x1, x2)	Time for x1 cycles at I/O line 13; x2*10mS time-out (The Model 8 uses I/O line 4 instead of I/O line 13)
PIN (x1 [,x2, . . .])	Value formed from specified input lines
RIGHT(str\$, x)	Return a string made up of the right-most x characters of str\$.
SIN (x)	Sine of x
SQR (x)	Square root of x
STR(expr)	Return a string representing the value of expr . The type of the conversion (float or integer) is determined by the type of the expression (as decided by the tokenizer).
TAN(x)	Tangent of x
TEMP (x)	Convert x to degrees C (times 100) for thermistor input
VAL(str\$)	Return the numeric value of str\$. Since the string could represent an integer or a float (determined at run-time), there is no way for the tokenizer to know the type of the return value at compile-time. VAL() is limited to variable assignments because the type of the variable can determine the type of the conversion.
VARPTR (v)	Address of variable v (not the Model 8)
VGET (x)	Value from EEPROM addr x (not the Model 4A or 5)
Digital I/O Control:	
PCLR x1 [,x2 . . .]	Clear specified I/O lines to low state
PSET x1 [,x2 . . .]	Set specified I/O lines to high state
PTOG x1 [,x2 . . .]	Toggle state of specified I/O line
SDI(x)	Return x shifted bits on I/O line 9; I/O line 3 latch, I/O line 5 clock (The Model 8 uses I/O line 7 instead of I/O line 9)
SDO x1, x2 SDO "s",[#n][,x][,\x][,{x,y}][:;]	Send x1 using x2 bits on I/O line 11,I/O line 2 latch,I/O line 5 clock Send 8-bit characters out serially as above (The Model 8 uses I/O line 8 instead of I/O line 11)
TONE x1, x2	Send x2 cycles, period x1*1.6276µSec out I/O line 12 (The Model 8 uses I/O line 6 instead of I/O line 12)

Table 5-3: TxBASIC Command Quick Reference (Continued)

Command	Description
Low Power and Time Commands:	
HALT	Dormant mode (Model 6 and 6-1m only)
HYB <x>	Wait out period in lowest power mode (6F only)
RTIME RTIME v	Read real-time clock to '?' array Read variable v to '?' array
SLEEP x	Sleep till x*10mS from last SLEEP command
STIME STIME v	Write '?' array data to real-time clock Write '?' array data to variable v
s = string x = expression v = variable f = burst storage form m = format { x,y } = datafile block\ x = 8-bit character given by x	

Strings in TxBASIC

String variables are denoted by using a \$ suffix on the name of the variable the first time it is used. The \$ suffix can be used in later references to the variable for consistency - it is just ignored. When a string variable is declared, 256 bytes are reserved for it. The first byte is the length of the string so a string cannot contain more than 255 characters. 256 bytes are always reserved for the string whether this space is used or not.

Like the integer and floating point variables, TxBASIC does not initialize its string variables. If the programmer does not initialize each string variable, it may contain from 0 to 255 characters of any type when it is accessed in the program.

String constants must be enclosed in double quotes. This is different from earlier versions where single or double quotes could be used. String constants can be up to 255 characters in length (although the editor only handles lines 256 characters long). String constants can be used as arguments to commands and functions and can be assigned to string variables.

String Functions in TxBASIC

String functions can be used just as any other function except for VAL(). The type of VAL conversion is determined by the variable type it is being assigned to so VAL() can only be used when assigning its value to a variable. All other functions can be used as a variable or argument in another expression. Of course, functions returning a string can only be used where a string is expected. Functions returning an integer can only be used when an integer is expected.

Table 5-4: TxBASIC String Commands

Name	Explanation
GETS(v)	Returns a string from the datafile starting at the datafile address in variable <i>var</i> . This assumes the string was stored in binary form.
INSTR([i,] str1\$, str2\$)	Return the position in <i>str1\$</i> at which <i>str2\$</i> is first found. Optionally start the search at position <i>i</i> in <i>str1\$</i> .
LEFT(str\$, n)	Return a string made up of the left-most <i>n</i> characters of <i>str\$</i> .
LEN(str\$)	Return the length of <i>str\$</i> . This can be a variable or constant.
MID(str\$, i, n)	Return a string made up of <i>n</i> characters starting at character <i>i</i> in <i>str\$</i> . The functions LEFT() and RIGHT() are really just special cases of MID().
RIGHT(str\$, n)	Return a string made up of the right-most <i>n</i> characters of <i>str\$</i> .
STR(expr)	Return a string representing the value of <i>expr</i> . The type of the conversion (float or integer) is determined by the type of the expression (as decided by the tokenizer).
VAL(str\$)	Return the numeric value of <i>str\$</i> . Since the string could represent an integer or a float (determined at run-time), there is no way for the tokenizer to know the type of the return value at compile-time. For now, we must limit the use of VAL() to variable assignments because the type of the variable can determine the type of the conversion. For instance, this is legal: afloat! = val(astring\$) but this is not: print val(astring\$)

Other String Handling Operations

- Input string to a variable using the INPUT command as in Input "Enter: " astring\$.
- String output/storing using the PRINT, STORE, SDO, USEND and CALL commands. Special instructions for STORE: the string is stored in a slightly different form in binary or ASCII forms. ASCII form is as expected but binary form stores the length byte first followed by the string. This allows the new GETS function to read strings from the datafile.
- String assignment (from other variables and from string constants) as in astring\$ = "Test Line".
- String concatenation using '+' operator. This is a legal operation:
astring\$ = "abc" + "defg" + "hij" where astring ends up as "abcdefghij"
- Special characters can be embedded in a string constant or character constant by preceding a pattern with the escape character '\'. This allows embedding a double quote character, a carriage return or any other character. For instance, to embed a double quote:

```
print "He said,\"I'm not sure\"."
```

```
results in:He said,"I'm not sure".
```

There is a list of special characters that can be used with the escape character:

\a = alarm, the BEL character = 07H

\b = backspace character = 08H

\f = formfeed character = 0CH

\n = newline, the line feed character = 0AH

\r = carriage return character = 0DH

\t = horizontal tab character = 09H

\v = vertical tab character = 0BH

\xcc = where cc is the hex value of the character to embed - there MUST be two characters for cc

\” = the double quote character = 22H

\’ = the single quote character = 27H

\\ = the backslash character = 5CH

Character Constants

Character constants have been added. You can now use ‘A’ as a synonym for &H41. There can be up to four characters enclosed in single quotes. The type of the character constant is a four-byte integer. Characters are shifted into the least significant byte of the integer as they are read with unused bytes set to zero. For instance, ‘A’ is a character constant equivalent to value 41 H. ‘AB’ is equivalent to 4142 H. ‘ABC’ is equivalent to 414243 H. ‘ABCD’ is equivalent to 41424344 H. Character constants can be used anywhere an integer value is used. These are especially useful when checking if a character from the UART is equal to one or more characters. For instance, if you got a character from the UART in variable ch:

```
if ch == ‘Y’ | ch == ‘y’ print “Yes”  
if ch == ‘N’ | ch == ‘n’ print “No”
```

Special characters can be embedded in a character constant by preceding a pattern with the escape character ‘\’. See the full explanation under “Other String Handling Operations”.

ABS

absolute value

Syntax: value = ABS(<x>)

Description: ABS returns the absolute value of the expression in parentheses. If the argument is an integer, the value returned is an integer. If a floating point argument is used, ABS is treated as a floating point function.

Example: **Write this program in TxTools:**

```
print abs(-7)
print abs(-7.0)
```

The following will be displayed when you run it in TxTools:

```
7
7.000000E0
```

Cautions: An integer expression passed to ABS must evaluate to a number between -2147483647 and 2147483647. Passing a larger or smaller value will cause the interpreter to stop execution and print "HOW?" error message.

ADLOOP start background conversion routine

NOTE: This command is not available for the Model 8. See the Burst2KSetup, BurstAD and BurstInfo extensions for a similar command for the Model 8.

Syntax: ADLOOP

Description: This macro command is an optimized command that can be used to make conversions to a circular buffer in the datafile as a background task. The task is controlled by writing to members of the ADLOOP array. This array is predefined by TxBASIC and contains eight variables. ADLOOP can only be started in the foreground because it is a background task itself (although not started with RUN). ADLOOP can only be used on Tattletales with the 12 bit A-D converter.

The following values are defined in the ADLOOP array:

ADLOOP(0)	non-zero to start (must still execute ADLOOP to begin), zero to stop
ADLOOP(1)	result of most recent conversion
ADLOOP(2)	next address to store to (on start-up, automatically set to ADLOOP(4))
ADLOOP(3)	channel to convert
ADLOOP(4)	start of ring buffer in the datafile
ADLOOP(5)	end of ring buffer in the datafile
ADLOOP(6)	zero, or start of assembly language routine to call before returning from ADLOOP interrupt (use to increment the channel for example).
ADLOOP(7)	The least significant 8 bits of this are ORed with the conversion value before storing.

The value in parentheses MUST be a constant from 0 to 7. As with all other TxBASIC variables, the ADLOOP array is not initialized on power-up or system reset. Before starting the routine (by setting variable ADLOOP(0) to a non-zero value and executing ADLOOP), ADLOOP(3) through ADLOOP(7) must be initialized. An example of this is given below.

Example: (indentation for clarity only except in ASM section)

< This first section must be executed to assign the equates for later use >

```
asm $
ADL    equ &H7C00 ; the ADLOOP parameters start at this addr
ADL_CH_B equ ADL+15 ; A-D channel number, adloop(3)
ADL_PAD_B equ ADL+31 ; LS byte fill value, adloop(7)
end
```

< This section not executed, it is loaded into General User Area 2 >

```
asm &H74C0
    ldaa ADL_CH_B ; get current channel
    inca      ; select next channel
    anda #7 ; just 0 to 7
    staa ADL_CH_B ; save it for ADLOOP to find
    staa ADL_PAD_B ; store channel number with data
    rti ; NOTICE THIS INSTRUCTION !
end
```

< Initialization of the ADLOOP parameters >

```

adloop(0) = 0           // stop if already acquiring
adloop(1) = 0           // clear conversion value
adloop(2) = 0           // first address is zero
adloop(3) = 0           // start with A-D channel 0
adloop(4) = 0           // datafile starts at 0
adloop(5) = 1000        // datafile ends at 1000
adloop(6) = &H74C0      // address of routine to select channels
adloop(7) = 0           // channel number will be ORed in with data
adloop(0) = 1           // start acquiring data
lastptr = 0             // init check value (checks when pointer wraps around)

```

< The acquisition loop >

```

adloop                // execute ADLOOP to actually start the acquisition
loop:
  if adloop(1)>32768 adloop(0) = 0: stop // stop if latest value>half scale
  if adloop(2) < lastptr print "wrapped" // wrapped to start of datafile
  lastptr = adloop(2) // update the wrap indicator
goto loop

```

Remarks: ADLOOP is not available for the Model 8 at this time. See the Burst2KSetup, BurstAD and BurstInfo extensions for a similar command for the Model 8.

Cautions:

1. Do not start another background task while ADLOOP is running and do not start ADLOOP if you already have a background task running. In later versions of TxBASIC, there will probably be a HOW error for this.
2. If you define an assembly language program with ADLOOP(6), it must finish with the RTI mnemonic (return from interrupt). The ADLOOP routine runs off an interrupt.
3. The example above shows the preferred way to pass an assembly routine's address to ADLOOP, but if you must write an assembly routine using the ASM \$ syntax to be used with ADLOOP:
 - a. Assign a TxBASIC label to the ASM \$ line. Remember, this must end with a colon. For example:

```

asmLabel:      asm $
                  <assembly code here>
                  end

```

- b. DO NOT EXECUTE THAT SECTION OF CODE because the assembly code ends with an RTI. You can only execute an RTI if that section of code was launched in response to an interrupt.

- c. Use this method to pass the address of the assembly routine to ADLOOP:

adloop(6) = labptr(asmLabel) + 4

There will be four bytes of header information between the ASM \$ line and the start of the actual code. That's the reason for the + 4 in the line above.

A number of commands will not respond well to changes in RATE and operation of ADLOOP.

Command	Failure
USEND, UGET	Neither of these expect the unusual interrupt rates or longer interrupt times of RATE and ADLOOP.
TONE	The maximum TONE frequency will be reduced by about a factor of 7 when ADLOOP is running.
COUNT, PERIOD	These maximum rates will be reduced as well.
CHAN, BURST	These will conflict with ADLOOP.

See also: RATE on [page 5-78](#).

AINT

round float down to integer

Syntax: value = AINT(<x>)

Description: AINT returns the next integer value less than the argument. The value is returned as a float. The argument must be a float. If it is an integer, it will be converted to float first.

Example: **Write this program in TxTools:**

```
inData! = 23.7
result! = aint( inData )
print "aint of ",#.5F, inData, " = ", result
result = aint( -inData )
print "aint of ",#.5F, -inData, " = ", result
```

The following will be displayed when you run it in TxTools:

```
aint of 23.70000 = 23.00000
aint of -23.70000 = -24.00000
```

Cautions: Remember, this function does not simply strip off the fractional part of the argument. Negative numbers return the next lower whole number!

ASFLT

interpret argument as float

Syntax: value = ASFLT(<x>)

Description: ASFLT is used to tell TxBASIC how to interpret data in an ambiguous situation. The most common use for this function is when retrieving floating point values from the datafile or EEPROM.

Example: **Write this program in TxTools:** (indentation for clarity only)

```
dfPoint = 0
fltValue! = 1.0
for i = 1 to 3
    store dfPoint, fltValue
    fltValue = fltValue * 10.0
next i
dfPoint = 0
for i = 1 to 3
    print #08H, get(dfPoint,#4)    //print the three values in binary
next i
print
dfPoint = 0
for i = 1 to 3
    print #1F, asflt(get(dfPoint,#4)) //print the three values as floating point
next i
```

The following will be displayed when you run it in TxTools:

```
3F800000
41200000
42C80000
1.000000
10.000000
100.000000
```

Cautions: This function does not convert data from one type to another (as in FIX, FLOAT or INT). It simply tells TxBASIC that the argument is a floating point value.

See also: GET on [page 5-42](#) and VGET on [page 5-110](#).

ASM

assemble to memory

NOTE: Must specify model with the Model command when using this command.

NOTE: This command is not available for the Model 8.

Syntax: ASM \$ or ASM <address>
 <code>
 <code>
 END

Description: Assembly language can be written directly using ASM. All 6303 instructions are supported in all addressing modes. Assembly starts when it encounters the ASM command and stops when it encounters the END command. This assembler allows the use of named labels and it can access TxBASiC variables by name.

If the optional <address> is included, the code is assembled starting at the specified address. If <address> is replaced with the \$ character, assembly is done in line and automatically called when reached.

Labels: Labels can be used in the assembly code for flow control and to define local variables. Labels **MUST** start in the first column. Labels can be up to 32 characters long and must begin with a letter or an underscore (_). The only valid characters in a label are upper and lower case characters, the numbers and underscore. The label name can be terminated with a colon (when the label is defined) but this is not necessary in the assembler. These labels are accessible to TxBASiC but TxBASiC labels are not accessible to the assembly code (although TxBASiC variables are).

Opcodes: The TxBASiC assembler recognizes all of the opcodes defined in the Hitachi 6301/6303 literature except for BCLR, BSET, BTGL, BTST, BHS and BLO (which are just pseudonyms for the AIM, OIM, EIM, TIM, BCC and BCS opcodes respectively). Opcodes must have at least one character of whitespace (space character or tab) in front of them on the line OR a label terminated with a colon.

In line assembly code (ASM \$)

This version allows you to install assembly language code that will be executed in line with the TxBASiC code and has the form:

```
<TxBASiC code>
ASM $                               (nothing else on this line, not even comments!)
<assembly code>
<assembly code>
...
<assembly code>
end
<TxBASiC CODE>
```

In the assembly section, everything after a semicolon and up to the end of the line, is considered a comment. The assembler does not recognize TxBASiC comments or REM commands. A more detailed explanation of this form of ASM is given in the TxBASiC Assembly language section earlier in the manual.

Notice that the first form of the ASM command \$ argument provides no way to initialize the A, B or X registers before entering the assembly code section. This can be done with the second form of ASM (address argument).

Assembly to an address (ASM <address>)

When the interpreter reaches this point in the program, it DOES NOT EXECUTE THE ASSEMBLY CODE. Instead it loads the code to the address specified by the ASM command until it finds the 'end' statement:

```

<TxBASIC code>
ASM <address>           (nothing else on this line, not even comments!)
<assembly code>
<assembly code>
...
<assembly code>
end
<TxBASIC CODE>

```

The A, B and X registers have a total of 32 bits. CALL initializes these on launch using '<input parameters>', and returns their values at exit in the '<optional output variable>'. In both cases the registers are packed the same way:

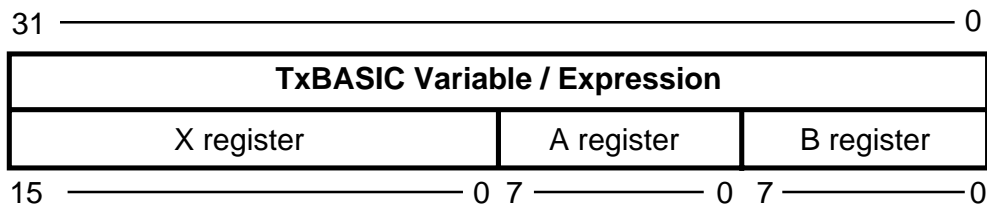


Figure 5-1: TxBASIC Variable Registers

The assembler automatically appends an RTS to the end of your code. If your assembly routine is launched by an interrupt you should end your code with an RTI. The assembler will append an RTS to this but it will not be executed.

A more detailed explanation of this form of ASM is given in the TxBASIC Assembly language section earlier in the manual.

Radix: Another thing to notice in these examples is the new methods of defining the number base of constants. You have these options IN THE ASSEMBLER ONLY in defining a constant 19 decimal as:

hexadecimal:	13H or &H13 or H'13 or \$13	<i>notice &H works as in TxBASIC</i>
octal:	23O or 23Q or Q'23 or @23	
binary:	10011B or B'10011 or %10011	
decimal:	19 or 19D or D'19	<i>decimal is the default number base</i>

Remarks: ASM is not available for the Model 7.

See Also: CALL on [page 5-19](#) and [“TxBASIC Assembly Language”](#) on page 4-28.

ATN

arctangent

Syntax: value = ATN(<x>)

Description: ATN returns the angle (in degrees) of the tangent expression in parentheses. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
tangent! = 1.0           // init arg (notice '!' means it's a float)
degrees! = 0.0         // just to force 'degrees' to be a float
for i = 1 to 5
  degrees = atn(tangent)
  print "The arctangent of ",#7.1F,tangent," is ",#6.3F,degrees
  tangent = tangent * 10.0
next i
```

The following will be displayed when you run it in TxTools:

```
The arctangent of    1.0 is 45.000
The arctangent of   10.0 is 82.289
The arctangent of  100.0 is 89.427
The arctangent of 1000.0 is 89.943
The arctangent of 10000.0 is 89.994
```

Remarks: The result of this function is in degrees, not radians.

See Also: COS on [page 5-25](#), SIN on [page 5-90](#) and TAN on [page 5-100](#).

BURST

Make block of A-D conversions to datafile

Syntax: BURST ptrvar, count, bytes
BURST <v>, <x> [,c]

Description: The four versions of BURST illustrated below demonstrate the different ways of storing a >8-bit number in the 8-bit wide datafile.

BURST <var>,<expr> stores the most significant 8 bits
 BURST <var>,<expr>,0 stores the least significant 8 bits
 BURST <var>,<expr>,1 stores the most significant 8 bits
 BURST <var>,<expr>,2 stores all bits using two bytes (left justified)

The result of the conversion is left justified to produce a 16-bit result, regardless of the number of bits in the converter; the result of a 12-bit converter is shifted left four bits, that of a 10-bit converter is shifted left six bits.

NOTE: Check for timing between samples at both crystal speeds.

Examples: (indentation for clarity only)

```

                                onerr finish           // exit when full
                                dfPoint = 0            // initialize file pointer
                                sleep 0               // sync up
measure:  sleep 1                          // wait 10 milliseconds
                                burst dfPoint,3       // first three channels
                                goto measure         // loop for next set
finish:   print "Done"                     // HERE WHEN DATA FILE IS FULL
wait:     sleep 100 : goto wait            // conserve power

```

Remarks: As with all data storage commands, the pointer variable is incremented after the completion of the command, leaving the variable pointing to the location immediately following the last location stored to.

Cautions: If an out-of-memory error occurs in the middle of a BURST command, the variable will be left with the value it had before the line was executed.

Specifying less than one channel or more than eight channels (eleven for the Model 5, four for the Model 6F) will generate a run-time error.

See Also: CHAN on [page 5-22](#), and GET on [page 5-42](#).

CALL

Call assembly language subroutine

NOTE: This command is not available for the Model 8.

Syntax: CALL addr, regs [, var]
 CALL <x1>, <x2> [, <v>]
 or
 CALL addr, "string" [, val][, \ascii val][{, df start, df end}][;]
 CALL <x1>, "<s>" [, <x2>][, \<x3>][{<x4>, <x5>}][;]

Description: CALL executes a user loaded assembly language subroutine.

Form 1 In the first form, the A, B, and X registers are loaded with the value in the "regs" expression on entry to the subroutine, and on exit, the optionally specified variable returns with the register contents. The packed format for register passing is shown below:

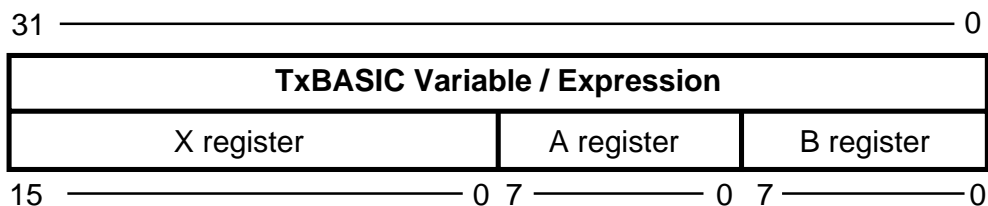


Figure 5-2: TxBASiC Variable Registers 2

Form 2 In the second form, strings, values, characters, and datafile blocks are passed to the CALL address <x1> one character at a time. The format is identical to that used in PRINT, except that the string <s> must follow the call address (it can be a null string). The other values, strings, characters and datafile blocks can be in any sequence. The assembly language routine at the CALL address is expected to have a procedure for disposing of the 8-bit characters (arriving in the 'A' register). This general form makes it possible to deal with any form of output device, such as a Centronics printer interface. For further details see [“TxBASiC Assembly Language”](#) on page 4-28.

NOTE: Both forms of CALL allow you to call an assembly subroutine by its label instead its numerical address. The example below shows how to use CALL with a label. The assembly routine must be defined before the CALL statement for the label to be valid.

These format specifiers, #D, #H, #B, #F, #S or #Q are considered ambiguous. Do D or H signify the radix (decimal or hexadecimal) or a variable field width? You can use variables for field width but not with those 12 names (upper or lower case). To get the radix form, use #1D, #1H etc. Refer to the PRINT command on [page 5-73](#) for additional information.

Example:

```
asm &h118
P5DDR equ #H'20 ; Data Direction Register for Port 5
PORT5 equ #H'15 ; Port 5 Data Register

setport ldaa #H'FF
        staa P5DDR ; set DDR to all output lines (all 1's)
        rts
output  stab PORT5 ; output least significant byte of argument
        rts
        end

print "This TxBASIC program outputs data to Parallel Port 5"
call setport,0
for i = 0 to 255
    call output,i
next i
```

Cautions:

When you call an assembly language program, you are leaving the warmth and safety of the TxBASIC programming environment. Obviously, the power to access all of the registers, ports and memory also affords a path to catastrophic program crashes which may be very difficult to diagnose.

Remarks:

CALL is not available for the Model 8.

See Also:

ASM on [page 5-15](#), [“TxBASIC Assembly Language”](#) on page 4-28 and PRINT on [page 5-73](#).

CBREAK

restart to label on Ctrl-C

Syntax: CBREAK [label]

Description: This command allows you to redirect the flow of your program when it receives a Control-C character. Normally, the program terminates and returns to the monitor. If you use the CBREAK command (followed by a label name) at the beginning of your program, it will vector to the label when it receives a Control-C character. To return to the normal Control-C action, use CBREAK with no argument.

Example: (indentation for clarity only)

```

// a SLOW version of a pseudo-ADLOOP
                cbreak getdata                // goto 'getdata' when Ctrl-C hit
                sleep 0
savedata:      dfpoint = 0
                for icount = 1 to 2000        // collect 2000 data pts
                    store dfpoint,#2,chan(1)
                    sleep 5
                next icount
                goto savedata                // reset datafile pointer to start
getdata:      print "Get ready to off-load"
                offld 0, dfpoint
                stop

```

This code fragment shows one use for CBREAK. When a Control-C character is received, the program will vector to the code at label 'getdata' and perform an OFFLOAD before stopping.

Remarks: The Control-C handler may be changed any number of times by executing CBREAK with different arguments.

As with ONERR , CBREAK must be executed to be effective. For this reason, you should put CBREAK near the beginning of the program.

TxBASIC will respond to a Control-C only while the foreground program is executing. A background task CANNOT be interrupted with a Control-C.

NOTE: The following applies to all models except the Model 8. You can disable CTRL-C breaks by writing a zero byte to address 9E hex (POKE &H9E,0). Re-enable CTRL-C breaks by writing a non-zero byte there (POKE &H9E, 1). A count of CTRL-C characters will continue to be updated at address 93 hex. Clear this before you re-enable break-outs (POKE &H93, 0).

Cautions: Watch out! Being too fancy with this command can cause a disaster. If the example were a subroutine, the error jump would wash out the return information, and its RETURN statement would cause an error.

NOTE: Will be cleared if "Boot from ROM" command is used by TxTools or if RAM has been found to be corrupted.

CHAN

get result of A-D conversion

Syntax: value = CHAN(<x>)

Description: The CHAN command returns a digital value corresponding to ratio of the voltage at the input channel specified by <x> to the converter's reference input. The result of the conversion is left justified to produce a 16-bit result, regardless of the number of bits in the converter; the result of a 12-bit converter is shifted left four bits, that of a 10-bit converter is shifted left six bits.

Example:

```
// **** CHAN EXAMPLE ****
for counter = 1 to 10
value = chan(0)
print #016B, value, 'B ', #04H, value, 'H ', #1D, value
next counter
```

Remarks: CHAN takes about 1ms to execute.

Cautions: Specifying a channel less than zero or greater than seven (ten for the Model 5, three for the Model 7) will generate a run-time error.

See Also: BURST on [page 5-18](#).

Ratiometric A-D's

Many Tattletales have their negative references tied to ground and their positive reference inputs tied to the converter's positive supply. This means that if you are measuring your sensor as a fraction of the reference input (a potentiometer, or a bridge), your conversion will give nice repeatable results.

The 12-bit A-D converters will work well with an externally applied reference that is 2.5V or greater (the converter's accuracy depends on the reference voltage and begins to deteriorate with reference inputs less than 2.5V).

Signal Conditioning of Analog Inputs

The analog inputs are designed to handle signals that range from 0 to the converter's Vcc. This full range ratiometric conversion is ideal for potentiometer inputs with the slider attached to the input and the two ends tied to Vsw and ground. Other sensors, such as strain gauges, need amplification before they can be attached to the converter input.

To minimize power consumption, the converters in the Models 2B and 2B-1M are not powered until about 125µSec before the conversion is started. This creates a problem as an amplifier would have only this much time to settle before the conversion is made. There are three ways to address this problem: make a fast amplifier, give the signal conditioning circuitry more time to settle, or leave the converter permanently powered.

The Model 5F, 5F-LCD and 6's converter is always powered, but it is forced into a low power mode by software after each use.

Adding More Setup Time using Assembly Language (not for Model 8)

All Tattletale models except the 5, 5F, 5F-LCD and 6 power switch the A-D converter. The time between power-up and the actual conversion is only 125 μ Sec (see the data sheet for the specific model for the actual number). This may be too little for some applications. The program below shows how this time can be extended. The first line of the program installs an assembly language routine that powers up the A-D converter. Line labeled 'convert' causes the program to be executed. See “[TxBASiC Assembly Language](#)” on page 4-28 and the ASM command on [page 5-15](#) for more details.

(indentation for clarity only except in ASM section)

```
asm &H118
    aim &H7F, &H17
    rts
    end
'Clearing MSB of port 6 of 6303 powers A-D
    sleep 0
convert:call &H118,0           // power up A-D
    sleep 3                   // 30mS delay
    print temp(chan(7))
    sleep 10                  // 100mS (total 130mS per loop)
goto convert
```

Another possibility is to just leave the A-D powered all the time. This can be done by setting the 'leave on' bit of the Configuration Byte. Refer to **Section 6 - Hardware and Interface Specifications** for more information.

How to add more A-D channels

More analog inputs can be added to a Tattletale by using some of its digital lines to multiplex the analog inputs already on the board. Adding a 74HC4051 in front of one input as shown in the diagram below multiplexes eight inputs into one input. Power the 4051 from the +V and ground lines so the multiplexer will be continuously powered. Three digital lines are lost to control the multiplexer.

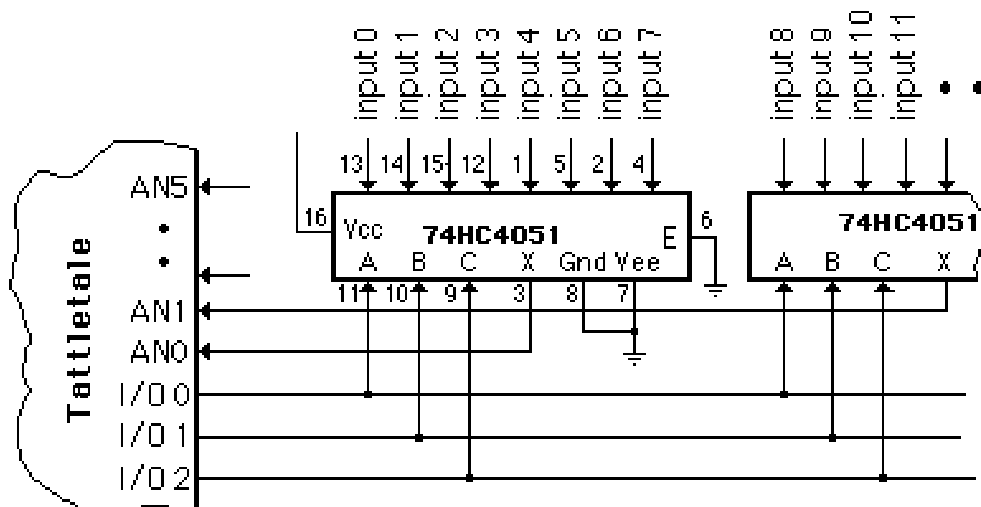


Figure 5-3: Schematic for Adding more A-D Channels

Before you can read a channel you must select the right input to the multiplexer by appropriately setting or clearing the I/O lines connected to the multiplexer.

```
pclr 0,1,2: x = chan(0)           // this reads one channel
pclr 1,2:pset 0: y = chan(0)     // this reads another
```

You can even automate the channel selection.

```
dim @(48)
for A = 0 to 47                  // read all 48 channels to the @ array
  gosub getChan                 // returns channel 'A' result in X
  @(A) = X : next A
stop
getChan:
  B = A : C = A%8 : D = A/8     // 'C' = mux addr, 'D' = converter chan
  for E=0 to 2                  // set the mux address one line at a time
    if B%2 <> 0 pset E : goto skip // set line if bit is high
    pclr E                      // otherwise clear it
  skip:
    B = B/2 : next E           // then do all three lines
    X = chan(D)                // make conversion
  return                        // return with result
```


COS

cosine

Syntax: value = COS(<x>)

Description: COS returns the cosine of the expression in parentheses. The argument must be degrees. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
degrees! = 0.0           // init arg (notice '!' means it's a float)
result! = 0.0           // just to force 'result' to be a float
for i = 1 to 6
  result = cos(degrees)
  print "The cosine of ",#5.1F,degrees," is ",#6.3F,result
  degrees = degrees + 72.0
next i
```

The following will be displayed when you run it in TxTools:

```
The cosine of    0.0  is  1.000
The cosine of   72.0  is  0.309
The cosine of  144.0  is -0.809
The cosine of  216.0  is -0.809
The cosine of  288.0  is  0.309
The cosine of  360.0  is  1.000
```

Remarks: Don't forget, the argument to this function is in degrees, not radians.

See Also: SIN on [page 5-90](#), TAN on [page 5-100](#) and ATN on [page 5-17](#).

COUNT

count positive edges at I/O line 13

NOTE: The Model 8 uses I/O line 4 for the COUNT command (instead of I/O line 13).

Syntax: cycles = COUNT(duration)
 cycles = COUNT(<x>) (count function)
 COUNT <v> (count command)

Description: There are two versions of COUNT. The count *function* counts the number of square wave cycles (positive edge is counted) appearing at I/O line 13 during a specified time. Specify the duration <x> in hundredths of seconds between 1 and 65535.

The count *command* works in the background incrementing variable <v> at every positive edge on I/O line 13. COUNT <v> starts the background counter and COUNT with no arguments disables the count interrupt. *The COUNT and PERIOD functions may not be used while the COUNT command is running.*

Examples: **Write this program in TxTools:**

```
for N = 2000 to 6000 step 2000
  print "Set up for ", N, " Hz, hit <cr>...";
  input "" A; // input to dummy var to wait
  print " reads ", count(100), " Hz"
next N
```

The following will be displayed when you run it in TxTools:

```
Set up for 2000 Hz, hit <cr>... reads 1998 Hz
Set up for 4000 Hz, hit <cr>... reads 4000 Hz
Set up for 6000 Hz, hit <cr>... reads 6003 Hz
```

Write this program in TxTools:

```
count backCount // start background count, store in backCount
SLEEP 0:SLEEP 2
A=backCount // save current count, counting continues
SLEEP 100
B=backCount:PRINT B-A // get second count, show difference
count // stop the background count
PRINT COUNT (100) // now count with function
```

The following will be displayed when you run it in TxTools:

```
1998
1997
```

Remarks: Triggering off the positive edge. In the same amount of time, the PERIOD function will return a more accurate measurement. COUNT has two advantages:

1. It can be simpler to use.
2. The number of transitions in 'duration' can be unknown.

Cautions: The maximum input frequency is about 30KHz (or 15KHz with a 4.9MHz crystal). Rates higher than that will return erroneous results. On the Model 8, COUNT will work at input frequencies > 100KHz.

Durations greater than 65535 or less than 0 will generate run-time errors.

See Also: PERIOD on [page 5-70](#).

DFREAD

read datafile from disk

NOTE: This command is for the Model 6, 6-1M and 6F only.

Syntax: DFREAD <x1>,<v>[,<x2>]

Description: This command reads the contents of the logical block <x1> on the disk to the datafile, replacing all data in the datafile. <x1> must evaluate to a number between 0 and the maximum storage capacity of the disk(s) in datafiles. The variable <v> will be set to zero if no error occurred and non-zero if there was an error. The optional <x2> must be either 0 or 1, and causes the corresponding half (only) of the disk file to be written to the corresponding half of datafile.

Example: This example will off-load a 20 MB disk if the receiving computer has an appropriately written receiving program.

```
dataFileEnd = 32767*7-1           // for 224K datafile, adjust for yours
for dfNumber = 1 to 92
  dfread dataFileNumber,errorNumber
  if errorNumber <> 0 print "Error on datafile ",dfNumber : stop
  offld 0,dataFileEnd
next dfNumber
```

This procedure is very slow as 20 megabytes must be transferred at 2000 bytes per second (this would take about 3 hours just for the transfer). For a Model 6, off-loading can be accomplished much more quickly (about 10 seconds per megabyte) using Onset's 'Direct-6' off-loading hardware (connects directly to a PC/AT or compatible).

Remarks: DFREAD takes about 10 to 19 seconds to execute, depending on disk capacity and datafile size.

Cautions: The hard disk takes about 1 to 3 seconds to spin down after a DFREAD is completed. A DFREAD or DFSAVE started before the spin down is completed will not be successful.

DFSAVE

store datafile to disk

NOTE: This command is for the Model 6, 6-1M and 6F only.

Syntax: DFSIZE <x1>,<v>[,<x2>]

Description: This command stores the contents of the datafile to the logical block <x1> on the disk, replacing all data in that block on the disk. <x1> must evaluate to a number between 0 and the maximum storage capacity of the disk(s) in datafiles. The variable <v> will be set to zero if no error occurred and non-zero if there was an error. The optional <x2> must be either 0 or 1, and causes the corresponding half (only) of the datafile to be written to the corresponding half of the disk's storage area. This allows you to split the datafile into two areas for ping-pong buffering.

Examples: This example fills a 20 MB of disk with measurements taken at 2400 meas/sec. Ping-pong buffering makes for no gaps in the measurement stream. This program is for the Model 6.

```

        adloop(0)=0 : adloop(1)=0
        adloop(3)=0 : adloop(4)=0
        rate 24                                // 2400 meas/sec
        adloop(5)=32768*7-1                    // whole df as buffer
        adloop(6)=0:adloop(7)=0
        adloop(0) = 1                          // enable ADLOOP to start
        adloop                                  // start ADLOOP
        for D=1 to 92                          // fill whole disk
loop1:  print #8,adloop(2),adloop(1)
        sleep 0:sleep 100
        if adloop(2)<16384*7-1 goto loop1
        dfsave D,X,0                          // store 1st half
        if X<>0 print 'DISK ERROR'
loop2:  print #8,adloop(2),adloop(1)
        sleep 0:sleep 100
        if adloop(2)>16384*7-1 goto loop2
        dfsave D,X,1                          // store 2nd half
        if X<>0 print 'DISK ERROR'
        next D                                 // do next df
        adloop(0)=0                          // stop ADLOOP

```

Cautions: The hard disk takes about 1 to 3 seconds to spin down after a DSAVE is completed. A DFREAD or DFSIZE started before the spin down is completed will not be successful.

DFSIZE

set datafile size

NOTE: This command is for the Model 4A and 5 only.

Syntax: DFSIZE <size>

Description: This command is used to set the size of the datafile in TxBASIC for the Model 5 Tattletale. The Model 4A and 5 powers-up with a default datafile size of zero bytes. Use this command to partition <size> bytes of RAM between datafile and TxBASIC variables. *Somewhere near the top of a Model 5 program must be a DFSIZE command. If you request more memory than is available, the tokenizer will give you what ever is left over from the program and variables. The PC version of TxTools will tell you how much datafile you actually have. This has not yet been added to the Macintosh version of TxTools.*

Example: dfsize 1024 ' make 1K datafile

Cautions: DFSIZE must be the first statement for a Model 5, and must not be included for any other model. The model will also have to be specified with the Model command.

DIM

dimension array

Syntax: DIM <label> (size [,size])

Description: This command is used to define an array. In TxBASiC, any legal variable name can be used as an array name as long as its size is defined first using this command. All array members take 4-bytes just like any TxBASiC variable (even floating point variables). DIM cannot be used with string variables.

DIM allows you to 'dimension' an array. You can optionally give an array up to two dimensions. It must appear in the program before any reference to members of the array. Attempts to access members outside the array's boundaries (as in Array1(21) or Array1(-1) in the example below) result in a 'HOW?' error.

Examples:

```
DIM Array1(20)      // make Array1 have 20 elements numbered 0 to 19
DIM Array2!(2)     // 2 floating point elements numbered 0 and 1
DIM Array3(100)   // make Array3 have 100 elements numbered 0 to 99
DIM Array4(10,6)  // array 4 has two dimensions, 0-9 and 0-5.
```

Remarks: References to the members of the Array1 defined above will look like Array1(0), Array1(1), Array1(2) ... Array1(19). Notice that the index starts at 0 and ends at 19, giving the same number of elements as the number in parentheses. After dimensioning an array in two dimensions, you can access the elements of the array using either two dimensions or as one longer dimension. For example:

```
DIM Array(5,7)
```

The array could be quickly filled like this:

```
For I = 0 to 34
  Array (I) = 0
Next I
```

Cautions: The @ array is not automatically available. It must be declared first. Once a name has been defined as an array, it cannot be used for a variable name and vice versa.

DISPLY**Display 4-character string on LCD**

NOTE: This command is for the Model 5F-LCD only.

Syntax:

```
DISPLY "s"[\,<x>][,#n][,val][,<x2>, <x3>][,;]
```

Description:

This command has the same syntax as print, but displays the data to the 5F-LCD's LCD. If more than four characters are requested, only the final four characters are displayed. The characters will be displayed until a different set are displayed. The characters that can be displayed are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, c, C, d, E, F, g, H, h, i, J, l, L, n, N, o, O, P, R, r, S, t, u, U, y, -, , ?, ., =, * (* is drawn as °).

```

ABCDEFGHIJKL MNOPQRST UVWXYZ
AbCdEF6H IJ L NOP rStU y
  c    h i    no  f  u
? * = 1 2 3 4 5 6 7 8 9 0 -
? ° = 1 2 3 4 5 6 7 8 9 0 -

```

Figure 5-4: 5F-LCD Characters that can be Displayed

NOTE:

The display has only seven segments plus a decimal point, and accordingly some characters are not displayable. The undisplayable characters will be skipped if an upper case or lower case equivalent cannot be substituted either.

See Also:

Scroll on [page 5-85](#).

DOFFLD

offload data from disk

NOTE: This command is for the Model 6, 6-1M and 6F only.

Syntax: DOFFLD <x1>,<x2>,<v>

Description: This command offloads data from the specified range of datafiles, starting at <x1> and progressing through <x2>, returning error information in variable <v>. The Model 6F will first look to see if the parallel interface is operating, and if it is, will offload the data through it. If the parallel interface is not operating, it will attempt to make an X-modem offload of the data through the serial port (at 19,200 baud).

Parallel offload requires that the Jason7 board be installed on the 6F, and that the appropriate parallel interface is made to a computer running Onset parallel offload software.

The serial offload requires the host computer to start an X-modem receive within one minute of the start of the execution of the DOFFLD command.

The variable <v> will return with value zero if the save is done without error, or a non-zero value if an error occurs.

DOFFLD can be either an immediate or program command.

Example: DOFFLD 1,39,X ;offload entire 20 Megabyte disk except DF 0.

NOTE: Offloading an entire 40 Megabyte drive using the DOFFLD command and the parallel interface and Onset host computer software takes than 15 minutes. A serial offload takes about five hours.

Error codes:

2	Track 0 on drive not found after power-up
16	Disk sector ID not found
17	Stopped while sending block of data to Host
33	Stopped while waiting for character from Host
64	Uncorrectable data error found on disk
65	Stopped while sending command information to Host
-	Any other codes: consult factory.

DONE

Enter oscillator stopped mode

NOTE: This command is for the Model 4A only.

Syntax: DONE

Description: This command puts the Model 4A into its lowest power mode where it has a typical power drain of 15 μ A. The only exits from this mode are disconnecting and re-connecting the main battery, connecting the two wakeup lines on the interface, or pressing the wakeup button on the interface cable. On exiting this mode, the Model 4A will start the program as though started with *RUN.

Examples:

```
X=0
ONERR label2 // Low power mode when the datafile is filled
SLEEP 0
label1: BURST X,5,2
        SLEEP 100
        GOTO label1
label2: DONE
```

Remarks: The DONE command is an effective way of preserving the Model 4A's program and data for extended periods without actually disconnecting the battery.

If the battery supply drops below 6.5V (by being discharged or disconnected), the board will drop directly into its low power state as though it had executed 'DONE'. The board will restart when the supply rises above about 7V.

Cautions: While in the dormant mode (after executing 'DONE'), all of the I/O lines are converted to inputs and are not asserted except by their pull-down resistors (I/O line 4 has a pull-up). Any hardware attached to the Model 4A should drop into its minimum power drain state when the lines are so asserted.

DSKON, DSKOFF**Power up disk, power down disk**

NOTE: This command is for the Model 6, 6-1M and 6F only.

Syntax: DSKON <v>, DSKOFF

Description: The DSKON command powers up the Tattletale's hard disk. If the power-up is successful the variable <v> will have a value 0; if unsuccessful it will be 10000. Subsequent DFREAD and DFSAVE commands will not power down the disk until DSKOFF is executed. These commands are designed to eliminate the spin-up and spin-down overhead time associated with the disk operations. Disk operations will be executed in roughly 3 seconds instead of the normal 10 seconds. Note however that while continuously powered the disk will draw about 200mA (from a 10V supply). The disk will not remain powered if DSKON is executed as an immediate command since a DSKOFF is executed (if the disk is powered) at the start of every immediate command.

The DSKOFF command has no error return variable. DSKOFF will take about 2 seconds to execute, allowing time for the disk to spin down. A DSKOFF command executed when the disk is

Example:

```
DSKON X
FOR A=1 TO 10
  DFSAVE A,X
NEXT A
DSKOFF
```

Cautions: A program that uses DSKON and DSKOFF should prevent the disk from being left powered if interrupted by a CTRL-C's vector to Line 100, or an ONERR vector.

Examples:

```
DSKON X           //This program will break to line 100
label1: GOTO label1 //When it sees a ctrl-C
label2: DSKOFF    // and turn the disk off, before stoping
STOP
*RUN
```

```
DSKON X
ONERR label3
:
:
label3: DSKOFF    // Turn the disk off, before stoping
STOP
```

DTOA

set dutycycle output (I/O line 12)

NOTE: This command is not available for the Model 8.

Syntax: DTOA <x>

Description: This sets the dutycycle of I/O line 12 approximating a D to A output. The output will be high $\langle x \rangle / 65536$ of the time and low $(65536 - \langle x \rangle) / 65536$ of the time. Acceptable values of the dutycycle are 750 to 64800 (out of 65536) values between 0 and 750 will produce 750/65536 output, and values between 64800 and 65536 will produce a 64800/65536 output. The command DTOA 0 disables the dutycycle output, leaving the line low. The dutycycle's fundamental frequency is 300Hz with a 4.9MHz crystal and 600Hz with a 9.8MHz crystal, and has a resolution of 16 counts out of 65535.

A typical smoothing circuit is shown in Figure 5-5. This circuit has a 0.1 second time-constant so the output will have about a 2% ripple, and a 0.3 second settling time.

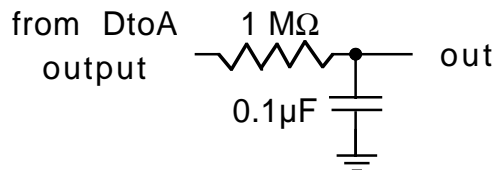


Figure 5-5: D–A Smoothing Circuit

EXP

raise e to a power

Syntax: value = EXP(<x>)

Description: EXP returns the base of the natural logarithms ($e = 2.71828\dots$) raised to the power of the expression in parentheses. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
arg! = 0.125                // initialize arg (notice '!' means it's a float)
result! = 0.0              // just to force 'result' to be a float
for i = 1 to 6
  result = exp(arg)
  print "e raised to ", #5.3F, arg, " is ", #6.3F, result
  arg = arg * 2.0
next i
```

The following will be displayed when you run it in TxTools:

```
e raised to 0.125 is 1.133
e raised to 0.250 is 1.284
e raised to 0.500 is 1.649
e raised to 1.000 is 2.718
e raised to 2.000 is 7.389
e raised to 4.000 is 54.598
```

Remarks: The range of input arguments is from -87.33654 to 88.72283 . Arguments greater than 88.72283 will result in an Overflow error (FPERR = 2) with the result equal to $+\text{Infinity}$. Arguments less than -87.33654 will result in an Underflow error (FPERR = 1) with the result equal to 0.0 . In both cases, execution is not stopped.

See Also: LOG on [page 5-59](#).

FIX

convert float to integer

Syntax: value = FIX(<x>)

Description: FIX returns the next integer value closer to zero than the argument. The value is returned as an integer. The argument must be a float. If it is an integer, it will be converted to float first.

Example: **Write this program in TxTools:**

```
inp! = 5.32987
result = fix(inp)
print "fix of ", #8.5F, inp, " = ", #1D, result
result = fix(-inp)
print "fix of ", #*.5F, -inp, " = ", #1D, result
```

The following will be displayed when you run it in TxTools:

```
fix of 5.32987 = 5
fix of -5.32987 = -5
```

Cautions: Arguments outside the range of Tattletale integers (-2147483648 to 2147483647) will result in a run time error.

See Also: FLOAT on [page 5-39](#) and INT on [page 5-52](#).

FLOAT

convert integer to float

Syntax: value = FLOAT(<x>)

Description: FLOAT returns the floating point representation of the argument. The argument must be an integer.

Example: **Write this program in TxTools:**

```
inp = 123
result! = float(inp)
print "float of ", inp, " = ", #5.1F, result
inp = -77
result = float(inp)
print "float of ", inp, " = ", #5.1F, result
```

The following will be displayed when you run it in TxTools:

```
float of 123 = 123.0
float of -77 = -77.0
```

Cautions: Single precision floating point has a precision of 24 bits while integers have a precision of 32 bits. Arguments outside the range of -16777215 to 16777215 will lose precision when converted to floating point. This Loss of Precision error (FPERR = 8) does not stop program execution.

A non-integer argument will cause an error in the tokenizer.

See Also: FIX on [page 5-38](#) and INT on [page 5-52](#).

FOR**for – next loop**

Syntax: FOR var = initial TO final [STEP inc] [statements]
 NEXT var
 FOR <v> = <x1> TO <x2> [STEP <x3>] [statements]
 NEXT <v>

Description: FOR loops provide one of four methods of looping available in TxBASIC. Here "var" can be any TxBASIC integer variable, and "initial", "final", and "inc" are integer expressions. The variable "var" will first be initialized to the value of the expression "initial", and then the section of code between the 'FOR' statement and the 'NEXT' statement will be repeated until "var" is greater than the value of the expression "final". After each pass, "var" will be increased by the value of the expression "inc". If STEP and "inc" are omitted, a step value of one is assumed. The limit ("final") and step ("inc") are evaluated and stored each time the loop is tested for continuation.

Examples: **Write this program in TxTools:**

```
//prints out the sequence 7,14,21,28,35,42,49
X = 7
FOR A = X TO X*X STEP X
  PRINT #4, A;
NEXT A
```

The following will be displayed when you run it in TxTools:

```
7 14 21 28 35 42 49
```

Write this program in TxTools:

```
//This example demonstrates the use of for loops for formatted print-outs
FOR A = 1 TO 3
  FOR B = 1 TO 5
    PRINT #5, B;
  NEXT B
  PRINT
NEXT A
```

The following will be displayed when you run it in TxTools:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Remarks: The test for continuation occurs at the beginning of the loop and the STEP and LIMIT expressions are evaluated each time this test is done. This slows the loop a bit but allows you to use a GOTO to exit the loop.

Because this structure stores nothing on the stack, you can nest these loops as deeply as you like. A GOTO can be used to exit any number of FOR loops.

Cautions: STEP may be positive or negative, but do not use STEP 0.

At this time, only integer variables and expressions can be used in the FOR loop specification.

See Also: WHILE on [page 5-112](#), REPEAT on [page 5-81](#) and GOTO on [page 5-45](#).

GET retrieve value from datafile

Syntax: value = GET(ptrvar [,#size])
value = GET(<v> [,#c])

Description: GET variable yields the value of the data in the datafile pointed to by the specified variable "ptrvar". As shown in the examples below, this can be used to format readouts as well as manipulate data in the logger's memory. GET optionally uses the specifier #1, #2, or #4 to specify whether the data should be retrieved as a single, double or quadruple byte integer. If no specifier is used, the 'byte' mode is assumed. The variable is incremented as each byte is read out.

Examples: **Write this program in TxTools:**

```
// This example prints out some of the data recorded in the 'BURST' example,
// 3-channels per line.
X = 0
FOR A = 1 TO 5 : FOR B = 1 TO 3
  PRINT #4, GET(X, #1);
NEXT B : PRINT : NEXT A
```

The following will be displayed when you run it in TxTools:

```
128 128 128
128 128 128
128 128 128
128 128 128
128 128 128
```

Write this program in TxTools: (indentation for clarity only)

```
dfPoint = 0 // start datafile pointer at 0
fltValue! = 1.0 // initialize our variable
for I = 0 to 20 //first fill EEPROM with n*n
  store dfPoint,fltValue*fltValue //store it (note – no ASFLT)
  fltValue = fltValue + 1.0 //update the variable
next I
dfPoint = 0 // reset datafile pointer to start
for I=0 to 20 //recover @(0) to @(20)
  //print square root(note use of ASFLT)
  print #6.1F,sqr(asflt(get(dfPoint,#4)));
  if I%10 = 9 print //print CR every ten items
next I
```

The following will be displayed when you run it in TxTools:

```
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0
20.0
```

See Also: ASFLT on [page 5-14](#), BURST on [page 5-18](#) and STORE on [page 5-97](#).

GETS

return a string from the datafile

Syntax: GETS(v)

Description: Returns a string from the datafile starting at the datafile address in variable *v*. This assumes the string was stored in binary form.

Examples:

```
astring$ = "This is a test string"  
x = 0  
store x, astring  
x = 0  
print gets (x)
```

Remarks: This function can be used anywhere a string variable or string constant would be used.

See Also: Binary forms of STORE on [page 5-97](#).

GOSUB go to subroutine, saving return address

Syntax: GOSUB label
GOSUB <x>

RETURN

Description: The GOSUB and RETURN commands allow you to use subroutines in TxBASIC. The label specifier can a line number or label. The RETURN statement signals the end of the subroutine. GOSUBs can be nested at least 20 deep.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```

//**** GOSUB EXAMPLE 1 ****
  GOSUB SUB1
  STOP

SUB1: PRINT "1st Subroutine"
      GOSUB SUB2
      RETURN

SUB2: PRINT "2nd Subroutine"
      RETURN

```

The following will be displayed when you run it in TxTools:

```

1st Subroutine
2nd Subroutine

```

Write this program in TxTools: (indentation for clarity only)

```

//**** GOSUB EXAMPLE 2 ****
Start: PRINT "BLOOD TYPE QUESTIONNAIRE"
      PRINT "Type '1' if O-neg, '2' if AB-pos, '3'";
      INPUT " if any other " X
      IF X =1 GOSUB DONOR
      IF X =2 GOSUB ACCEPTER
      IF X =3 GOSUB MAJORITY
      GOTO Start
      STOP

DONOR:  PRINT "You are a universal donor" : RETURN
ACCEPTER: PRINT "You are a universal accepter" : RETURN
MAJORITY: PRINT "You are in the majority!" : RETURN

```

The following will be displayed when you run it in TxTools:

```

BLOOD TYPE QUESTIONNAIRE
Type '1' if O-neg, '2' if AB-pos, '3' if any other 4
BLOOD TYPE QUESTIONNAIRE
Type '1' if O-neg, '2' if AB-pos, '3' if any other 2
You are a universal accepter

```

GOTO

go to label

Syntax: GOTO label
GOTO <x>

Description: GOTO causes an unconditional transfer of the program to the specified label. The label specifier can be a line number or label.

Examples:

```
***** GOTO EXAMPLE 1 *****  
    REM simple unconditional jump  
    GOTO SKIP  
    PRINT "you won't see this"  
SKIP: PRINT "you will see this"
```

Examples:

```
***** GOTO EXAMPLE 2 *****  
// Example of exiting a FOR LOOP with a GOTO command  
X = 0  
For I = 1 to 100000  
    STORE X, CHAN(3)  
    CALL 8HFFD9, 0, CHAR  
    IF CHAR <> 0 GOTO EXIT  
NEXT I  
PRINT "Normal end of loop"  
STOP  
EXIT: PRINT "Operator Terminated Loop"  
STOP
```

Remarks: GOTO can be used to exit any number of nested FOR, WHILE and REPEAT loops.

See Also: WHILE on [page 5-112](#), REPEAT on [page 5-81](#) and FOR on [page 5-40](#).

HALT

low power mode

NOTE: This command applies to the Model 6 only.

Syntax: HALT

Description: This command puts the Model 6 into its lowest power mode where it has a typical power drain of 500µA. The only exits from this mode are disconnecting and re-connecting the main battery, and pulling the RESET line low momentarily. On exiting this mode, the Model 6 will start the program as though started with Alt-B, or continue with the instruction that follows 'HALT' depending on the setting of a bit in the configuration bytes in EEPROM (see "**Configuration Byte Information**" in Section 6 for details about the configuration bytes).

Examples:

```

dfPoint = 0
onerr finish // low power mode when datafile filled
sleep 0
loop: burst dfPoint,5,2
      sleep 100
      goto loop
finish: halt

```

Remarks: The Model 6, by itself has no way of exiting the HALT command. To make use of HALT (in a way other than minimizing the current drain at the end of a deployment), you will need to add external hardware to wake up the Model 6. While HALTed all clocks on the Model 6 are stopped and the '?' variable stops incrementing.

While executing the HALT on the Model 6, all of the I/O lines revert to inputs, and should be held to CMOS logic levels by the pull-up/down resistors (except for I/O4 which has a pull-up). At the end of the HALT command, the lines revert to the states they were in before the command was executed if the Model 6 is in 'Halt continue mode'. In a Model 6 exiting HALT in 'HALT restart' mode, all lines become inputs and the user's BASIC program is restarted. See "Cautions" under the "HALT" command earlier in this manual.

A "pseudo-HALT" is available for the Model 2B. Use the command POKE(&H14,&H80). See Section 6 for details.

Cautions: While in the dormant mode (after executing 'HALT'), all of the I/O lines are converted to inputs and are not asserted except by their pull-down resistors (I/O line 4 has a pull-up). Any hardware attached to the Model 6 should drop into its minimum power drain state when the lines are so asserted. In order to make use of the low power HALT state you must provide terminations for several of the Model 6's lines:

Line	Termination	Remarks
-WR	10K to +5	
-RD,A0-A3	100K to +5	
I/O lines	1M to +5 or gnd	I/O 4 already terminated

* Note that in REV C of the Model 6 and beyond all -WR, -RD, A0, A1, A2, and A3 are all terminated appropriately, but the I/O lines are not (except I/O4).

See Also: Refer to the "**Configuration Information**" in Section 6 for additional information.

HYB

Wait out period in lowest power mode

NOTE: This command is for the Models 6F and 8 only.

Syntax: HYB <x>

Description: This command puts the Tattletale into its lowest power mode where it has a typical power drain of about 100 μ A (running from a 10V supply), but with a scheduled wakeup. <x> defines the wakeup in seconds from the previous exit from HYB, where $x < 32768$. Like SLEEP, there is a HYB 0 to define the first incidence of HYB. After executing HYB, the Tattletale wakes every ten seconds and checks the serial line for a break. If the serial line is in the spacing state (called the BREAK condition in the RS-232 specifications), the Tattletale will exit HYB as though it had received a CTRL-C. If the HYB period has expired without detecting a break character, the Tattletale will continue its program with the next instruction. The Model 8 will reduce its operating voltage to 3.0 V unless the HybAt3V extension was used to disable this.

In order to minimize its current drain, during HYB the 6F reduces its operating voltage to 4.0V. The Model 8 reduces its operating voltage to 3.0V.

Example:

```
HYB 0
label1: PRINT "HI"
HYB 25           // LOWEST POWER MODE FOR 25 SECONDS.
GOTO label1
```

Remarks: Upon exiting HYB mode the ? variable will be reset to 0 and the Model 6F will be running at RATE 1. All Model 6F I/O lines will be restored to the state they had before HYB was executed. The Model 8 will continue to run at the RATE it was set to before going into HYB.

Cautions: **For Model 6F:** While in the dormant mode (after executing 'HYB'), all of the I/O lines are converted to inputs and are not asserted except by their pull-down resistors (I/O line 4 has a pull-up). Any hardware attached to the Model 6F should drop into its minimum power drain state when the lines are so asserted. In order to make use of the low power HYB state you must provide terminations for several of the Model 6F's lines:

For Model 8: The Model 8 shuts down all subsystems (QSPI, TPU, SCI, the oscillator etc.) which are revived only by the scheduled wakeup or the break condition.

Line	Termination	Remarks
I/O lines	1M to +5 or gnd	I/O 4 already terminated

See Also: HybAt3V and HybCheckForBREAK extensions for the Model 8.

IF branch on result of comparison

Syntax: IF expression command
IF <x> command

Description: IF allows you conditional control of your program. If expression is true (does not evaluate to 0), command is executed; if expression is false (evaluates to 0), command is skipped and the program continues with the next line. If one of the operands of the comparison is a floating point value and the other is an integer, the integer is temporarily converted to a float and then the operands are compared as floating pointing values.

Examples: **Write this program in TxTools:**

```
//***** IF example 1 *****  
for A = 1 TO 200  
    if A = 100 gosub Special  
next A
```

```
Special:  
print "A = 100! Isn't that special?"  
return
```

The following will be displayed when you run it in TxTools:

A = 100! Isn't that special?

Write this program in TxTools:

```
//***** IF example 2. Same thing, another way *****  
for A = 1 TO 200  
    if A = 100 PRINT "A = 100. Never mind."  
next A  
stop
```

The following will be displayed when you run it in TxTools:

A = 100. Never mind.

Remarks: In the examples above, you could have used: If A = 100 | B = 100
In the examples above, you could have also used: If A > 99 & A < 101

See Also: IFF on [page 5-49](#) and [“Relational Operators”](#) on page 4-5.

IFF**branch on result of comparison**

Syntax:

```

IFF <x>
...command...
... block 1...
ELSE
...command...
...block 2...
ENDIF

```

Description: IFF allows you conditional control of your program. If the 'expression' is true (does not evaluate to 0), command block 1 is executed otherwise, if the ELSE block exists, command block 2 is executed. The ELSE block is optional but the ENDIF is not. If one of the operands of the comparison is a floating point value, the integer is treated like a float and the operands are compared as floating point values.

Examples: (indentation for clarity only)

```

// Example 1
input aValue
iff aValue > 100           // execute command block 1 if aValue > 100
  print "The value is greater than 100, but that's not allowed!"
  aValue = 100
else                       // otherwise, execute command block 2
  print "The value is NOT greater than 100 and that's OK"
  aValue = aValue + 3
endif                       // this defines the end of command block 2

```

```

// Example 2
for i = 1 to 200
  iff aValue = 76         // execute code from here to endif if true
    print "Go tell Prof. Hill all the trombones are here"
    gosub setFrontRow
    tbonesReady = 1
  endif                   // this defines the end of command block
  iff aValue = 110       // execute code from here to endif if true
    print "Let Prof. Hill know the cornets are here, too"
    gosub setSecondRow
    cornetsReady = 1
  endif                   // this defines the end of command block
next i

```

Remarks: Notice that a second 'F' is necessary in the name of this command. This helps the tokenizer distinguish between the old IF command (in which all statements must be on one line and doesn't allow an ELSE command block) and the new, more powerful IFF.

In the examples above, you could have used: If A = 100 | B = 100

In the examples above, you could have also used: If A > 75 & A < 77

See Also: IF on [page 5-48](#) and **“Relational Operators”** on page 4-5.

INPUT

get value from console

Syntax: INPUT ["prompt"] var[;] [,["prompt"] var[;] ...]
 INPUT ["<s1>"] <v1> [;] [,["<s2>"] <v2>[;]...]

Description: INPUT allows you to set a variable from the terminal. You can use a string (in single or double quotes) as a prompt. A default prompt of the variable name is used if you don't include one. If you don't want a prompting string, use a zero-length string (quotes with no intervening characters). Floating point input can be in fixed point or scientific notation.

Example: **Write this program in TxTools:**

```
input floatVar!
input intVar
input "Type a floating point value-> "test!
input "Type an integer value-> "number
print
print #10.3F,floatVar,#10D,intVar,#12.6S,test,#10D,number
```

The following will be displayed when you run it in TxTools: (typed responses in bold face)

```
floatVar:9.351e3
intVar:54321
Type a floating point value-> 134.55
Type an integer value-> 999999999 <too large for an integer>
? 999999999

9351.000 54321 1.345500E2 999999999
```

Remarks: Entering a carriage return alone in response to an input command assigns zero to the variable.

A trailing semicolon after the variable specifier causes the input command to inhibit echoing the terminating carriage return.

Entering a Control-C during an input leaves the variable unchanged.

The maximum number of characters that can be entered for an integer value is 15. The maximum number of characters for a floating point entry is 26.

Cautions: The expression entered from the terminal is evaluated after each prompt. If an integer value outside the range -2147483648 to 2147483647 is input, a '?' is displayed to request corrected input.

A floating point input between $-1.175494E-38$ and $+1.175494E-38$ assigns zero to the variable and, if the input is not exactly zero, sets the FPERR variable to indicate an underflow error. A value of +Infinity is assigned to the variable if the input is greater than $3.402823E+38$. A value of -Infinity is assigned to the variable if the input is less than $-3.402823E+38$. The value of FPERR will be updated to show an overflow occurred for either infinite result. If a floating point input cannot be evaluated, a '?' is displayed to request corrected input.

INSTR

returns the strings position

Syntax: INSTR([*x*,] *str1*\$, *str2*\$)

Description: Return the position in *str1*\$ at which *str2*\$ is first found. Optionally start the search at position *x* in *str1*\$.

Examples:

```
astring$ = "This is a needle in a haystack"  
search$ = "needle"  
Offset = instr (astring, search)  
if offset <> 0 print mid(astring, offset, len(search))  
if offset = 0 print "String not found"
```

Remarks: This function does not return a string. It returns an integer offset that can be used on string STR1\$.

See Also: LEFT on [page 5-56](#), RIGHT on [page 5-82](#), MID on [page 5-61](#), LEN on [page 5-57](#)

INT

convert float to integer

Syntax: value = INT(<x>)

Description: INT returns the next integer value less than the argument. The value is returned as an integer. The argument must be a float. If it is an integer, it will be converted to float first.

Example: Write this program in TxTools:

```
inp! = 5.32987
print "int of ", #8.5F, inp, " = ", #1D, int(inp)
print "int of ", #8.5F, -inp, " = ", #1D, int(-inp)
```

The following will be displayed when you run it in TxTools:

```
int of 5.32987 = 5
int of -5.32987 = -6
```

Cautions: Arguments outside the range of Tattletale integers (-2147483648 to 2147483647) will result in a run time error.

See Also: FIX on [page 5-38](#) and FLOAT on [page 5-39](#).

ITEXT

bring string from console to datafile

Syntax: ITEXT ptrvar[,\] [, timeout]
ITEXT <v> [,\<x1>][,\<x2>]

Description: The ITEXT command enters data directly to the logger's datafile. The specified variable is used as a pointer. A single line can then be loaded over the main UART. All characters including the terminating carriage return are then stored using the pointer, which is updated after the data is stored.

The default terminator for ITEXT is a carriage return. The terminator can be changed by following the pointer variable with a comma, backslash and then the ASCII value of the desired terminator.

ITEXT also has an optional time-out. If a time-out is specified and exceeded between the receipt of any two characters, ITEXT will bail out before the terminator. Time-out values of 1 to 65535 are allowed, specifying the time-out in 1/100ths of a second. time-out values outside of this range will evoke the 'How?' complaint.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```

***** SUPER-DUPER LOGGER PROGRAM *****
      GOSUB getheader      // get header info
//      \ super duper logger
//      / main body
      STOP

getheader: /*** INPUT HEADER INFO SUBROUTINE ***
      X = 2                // point to start + length
      PRINT "Deployment No. "; : ITEXT X
      PRINT "Instrument ID "; : ITEXT X
      PRINT "Chief Scientist "; : ITEXT X
      Z = 0 // point to length word
      STORE Z, #2, X      // header skip len.
      RETURN

```

The following will be displayed when you run it in TxTools:

```

Deployment No. 112
Instrument ID TT 302
Chief Scientist J. Williams

```

With Timeout: The hardware UART is buffered. If you want the time-out to specify the maximum time for receipt of the entire block instead of the interval between characters, specify the time in a SLEEP command and use the minimum interval in the ITEXT command as shown below.

Write this program in TxTools:

```
X=0: ITEXT X // Load periods and CR
X=0: OTEXT X // Show it
X=0: SLEEP 0: SLEEP 299: ITEXT X,1
X=0: OTEXT X // Show what was received
```

The following will be displayed when you run it in TxTools:

```
.....
.....
12345
```

Timeout = 0: If the time-out value is zero, one (and only one) character is brought in if one is ready. If none is ready, ITEXT will not wait for one. You can tell if a character has been brought in by checking if the pointer variable has moved.

Remarks: As with all data storage commands, the pointer variable is incremented after the completion of the command, leaving the variable pointing to the location immediately following the last location stored to.

Cautions: This command affords one of the few opportunities for the machine to appear to lock up and not respond to a CTRL-C, since it requires a carriage return to stop loading the string first.

If an out-of-memory error occurs in the middle of an ITEXT, the variable will be left with its value before the line was executed.

The time-out version of ITEXT does not echo characters, nor does it allow editing using backspace. This version is intended for communications with another machine.

The non-time-out version of ITEXT has a 240 character limit. Any characters after this will just be ignored until the terminator is detected, ending the command.

See Also: XMIT on [page 5-113](#) which provides another method of not echoing input characters.

LABPTR

return address of label

NOTE: This command is not available for the Model 8.

Syntax: LABPTR (<label>)

Description: This is a function that returns the absolute physical address of the label named in parentheses. The label must be defined somewhere else in the program.

Examples: (indentation for clarity only)

```
start_prog:
  print "square of values from 0 to 9"
  for a = 0 to 9
    b = a * a
    print #5, a, b
  next a
  print "last b value bytes: ";
  print #02H, peek(varptr(b)), " ", peek(varptr(b)+1), " ";
  print #02H, peek(varptr(b)+2), " ", peek(varptr(b)+3)
  print "program starts at ", #04H, labptr(start_prog)
  print "program ends at ", #04H, labptr(end_prog)
end_prog:
```

Remarks: This function does not work on the Model 8.
The model must be specified with the Model command to use this command.

See Also: VARPTR on [page 5-109](#).

LEFT

return left most characters

Syntax: LEFT(str\$, x)

Description: Returns a string made up of the left-most *x* characters of *str\$*.

Examples:
astring\$ = "This is not very important"
print left (astring, 17)

Remarks: This function can be used anywhere a string variable or string constant would be used.

See Also: RIGHT on [page 5-82](#), MID on [page 5-61](#), LEN on [page 5-57](#)

LEN**return the length of str\$****Syntax:** LEN(str\$)**Description:** Return the length of *str\$*. This can be a variable or a constant.**Examples:**
input "Enter up to 255 characters then CR: ", astring\$
print "You entered ", len(astring), " characters"
print astring**Remarks:** This function returns an integer: not a string**See Also:** INSTR on [page 5-51](#), INPUT on [page 5-50](#)

LET

assign value to variable

Syntax: [LET] var = expression [,var = expression...]
[LET]<v1>=<x1> [,<v2>=<x2>...]

Description: LET assigns a value to a variable. The expression is evaluated, and if the result is to be assigned to an integer variable and is not in the range -2147483648 to 2147483647 a 'HOW' complaint is shown. All floating point assignments are allowed except that out of range constants are errors in the tokenizer.

Examples: var1 = 1
var2! = 21.234e21
let var3 = 123
let Z = 1, R = 17, last! = 100.0

Remarks: The 'LET' can be omitted, reducing the command to an equation.

LOG

natural logarithm

Syntax: value = LOG(<x>)

Description: LOG returns natural logarithm of the expression in parentheses. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
arg! = 0.125 // initialize arg (notice '!' means it's a float)
result! = 0.0 // just to force 'result' to be a float
for i = 1 to 7
  result = log(arg)
  print "natural log of ", #5.3F, arg, " is ", #6.3F, result
  arg = arg * 2.0
next i
```

The following will be displayed when you run it in TxTools:

```
natural log of 0.125 is -2.079
natural log of 0.250 is -1.386
natural log of 0.500 is -0.693
natural log of 1.000 is 0.000
natural log of 2.000 is 0.693
natural log of 4.000 is 1.386
natural log of 8.000 is 2.079
```

Remarks: Arguments less than or equal to zero will result in a Not-a-Number error (FPERR = 4) but execution will continue. The result will be NaN.

See Also: EXP on [page 5-37](#) and LOG10 on [page 5-60](#).

LOG10

common logarithm

Syntax: value = LOG10(<x>)

Description: LOG10 returns the common logarithm of the expression in parentheses. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
arg! = 0.125 // initialize arg (notice '!' means it's a float)
result! = 0.0 // just to force 'result' to be a float
for i = 1 to 7
  result = log10(arg)
  print "common log of ", #5.3F, arg, " is ", #6.3F, result
  arg = arg * 2.0
next i
```

The following will be displayed when you run it in TxTools:

```
common log of 0.125 is -0.903
common log of 0.250 is -0.602
common log of 0.500 is -0.301
common log of 1.000 is 0.000
common log of 2.000 is 0.301
common log of 4.000 is 0.602
common log of 8.000 is 0.903
```

Remarks: Arguments less than or equal to zero will result in a Not-a-Number error (FPERR = 4) but execution will continue. The result is NaN.

See Also: LOG on [page 5-59](#).

MID return a string made up of *x* characters

Syntax: MID(str\$, x1, x2)

Description: Return a string made up of *x2* characters starting at character *x1* in *str\$*.

Examples:

```
offset = 1
astring$ = "One two three four"
while 1
    term = instr(offset, astring, " ")
    iff term = 0
        print mid(astring, offset, len(astring)+1—offset)
        print "Done"
        stop
    end if
    print mid(astring, offset, term—offset)
    offset = term + 1
wend
```

Remarks: This function can be used anywhere a string variable or string constant would be used.

See Also: INSTR on [page 5-51](#), LEFT on [page 5-56](#), RIGHT on [page 5-82](#) and LEN on [page 5-57](#)

MODEL

set Tattletale model type

Syntax: model <x>

Description: There are a number of commands where the TxTools tokenizer must know which Tattletale model the program is intended for. Refer to Table 5-5 for the model command format for each Tattletale model. These are the ASM and LABPTR commands and the DFSIZE, PIN, PSET, PCLR or PTOG if used with constant arguments.

Examples: To define the model for a Model 2B, for example, use this syntax near the beginning of your program:

```
model 230
```

NOTE: If you intend the program for different models, you can use multiple MODEL commands as in:

```
model 600 : model 610 : model 230
```

Remarks: Certain models are mutually exclusive. You will get a tokenizer error if you choose incorrectly. You must use the Model command for any Model 8 programs.

Table 5-5: TxBASIC Model Command Formats

Tattletale Model	TxBASIC Model Command Format
2A	model 210
2A-32	model 220
2B	model 230
2B-1M	model 240
3	model 300
4A	model 400
5	model 500
5F	model 510
5F-LCD	model 520
6	model 600
6-1M	model 610
6F	model 620
8	model 800

OFFLD

start X-modem transfer of datafile

Syntax: OFFLD start, end, timeout, error variable
 OFFLD <x1>,<x2>[,<x3>,<v>]

Description: The OFFLD command off-loads data from the logger using XMODEM protocol. The expressions 'start' and 'end' are evaluated, and the contents of the datafile locations 'start' to 'end' (inclusive) are sent. 'End' must be greater than 'start' and both must be in the range of zero to (use DFMAX Read-only value to get maximum value for 'End' argument):

98303 Model 5F-128 or 5F-LCD-128
 229375 Model 2B or 6
 491519 Model 5F-512P or 5F-LCD-512P
 1015807 Model 2B-1M or 6-1M

or the last location in the datafile for the Model 4A, 5 or 8. This will make a byte-for-byte copy to your computer's disk of that section of the Tattletale's datafile.

Data is sent out using XMODEM protocol so the number of bytes transmitted will always be a multiple of 128 bytes. If the number specified is not an even multiple of 128, zeros are used as fill bytes. Your disk file length therefore will be a multiple of 128 bytes.

OFFLD waits to receive a NAK (Ctrl-U) from the XMODEM receiving program before sending the data. The data transmission can be aborted by a Ctrl-X. The OFFLD command will abort if a NAK is not received within 60 seconds of the Tattletale's 'WAITING FOR NAK' prompt, if no time-out is specified. If the NAK is not received in time, the Tattletale will print 'NAK NOT FOUND BEFORE TIMEOUT'. If a time-out (timeout * 10 ms) and variable are specified, the 'WAITING FOR NAK', and the 'NAK NOT FOUND' prompts will be omitted.

If a time-out and error variable are specified at the completion of the off-load, the specified variable will contain:

First specified datafile location	<i>if no transfer at all occurred</i>
Last transferred location	<i>if partial transfer</i>
Last transferred location + 1	<i>if successful transfer</i>
-(Last transferred location+1)	<i>if all data sent, but final block's</i>
<i>EOT not acknowledged</i>	

Examples: (indentation for clarity only)

```
sleep 0
dfpoint = 0 // init datafile pointer
timeout = 7500 // OFFLD set-up time-out will be 77 sec
for icount = 1 to 2000 // collect 2000 data pts
store dfpoint,#2,chan(1) // collect and store data
sleep 5 // 50 milliSec between samples
next icount
print "You have ",timeout/100," seconds to prepare off-load..."
```

```

offld 0, dfpoint-1, timeout, errVar
iff (abs(errVar) <> dfPoint)
  print "Error: last transferred location = ", errVar
else
  print "Off-load succesful"
endif
stop

```

Remarks: The XMODEM protocol, sometimes called Christensen protocol, transfers data from the Tattletale when the OFFFLD command is used. This transfer protocol is supported by many computer programs. For those who wish to know the details of the transfer protocol we include a description below. Note that data is always sent in 128-byte blocks. If the total amount of data sent is not an even multiple of 128 bytes, it is filled with zeros to the next full block.

Definitions:

<soh>	01H	(Ctrl-A)	first character of data block
<eot>	04H	(Ctrl-D)	end of transmission character
<ack>	06H	(Ctrl-F)	acknowledge: data received ok
<nak>	15H	(Ctrl-U)	not acknowledge, or start offload
<can>	18H	(Ctrl-X)	cancel transmission

Message Block Protocol

Each block of data transferred will look like:

```
<soh><blk #><255-blk#><---128 data bytes---><cksum>
```

```
<soh>          = 01H
```

```
<blk#>         = hexadecimal number starting with 01, increments by 1 after each
                  block, rolls over from FF to 00 after the 256th block
```

```
<255-blk#>     = one's complement of <blk#>
```

```
<cksum>        = one-byte sum of data bytes only, carries ignored.
```

Data Flow Example

Tattletale transmissions are shown in plain type; computer transmissions are shown in bold face type. Three blocks are transmitted in the example below.

```

<nak>          starts offload
<soh>, 01, FE, <128 bytes>, <cksum>          first data block sent
<ack>          first block received ok
<soh>, 02, FD, <128 bytes>, <cksum>          second data block sent
<nak>          cksum didn't check!
<soh>, 02, FD, <128 bytes>, <cksum>          second data block re-sent
<ack>          second block received ok
<soh>, 03, FC, <128 bytes data>, <cksum>      third data block sent
<ack>          third block received ok
<eot>          that's the last block
<ack>          received end of transmission

```

If a <can> is received after any block is transmitted, it will abort the off-load. Any character other than an <ack>, <nak> or <can> will be ignored.

Cautions: *The Tattletale uses a checksum (not CRC) for block verification in ONLOAD, OFFFLD and REMIND. If your communications program can only use CRC, the transfer will not be successful!*

ONERR

go to line number on error

Syntax: ONERR [label [,var]]
ONERR [<x> [,<v>]]

Description: ONERR directs the Tattletale to jump to the specified label if a run-time error occurs instead of printing an error message.

Errors are normally flagged as they occur with a 'HOW' comment. If an ONERR line is encountered during execution, the error printout will be skipped and execution will continue at the 'label'. This allows emergency shutdown or recovery from a program error encountered in the field. To return to the normal error action, use ONERR with no argument.

When the ONERR branch is made, the program loses all information about previous GOSUBs.

Example: **Write this program in TxTools:** (indentation for clarity only)

```

    ONERR MID
    X=1
LOOP1: X=X*2 : GOTO LOOP1 // find something too big
MIDDLE: A=X : ONERR LOOP2
LOOP2: A=A/2
    IF A=0 PRINT "MAX INTEGER = ",X : STOP
    X=X+A : GOTO LOOP2

```

The following will be displayed when you run it in TxTools:

MAX INTEGER = 2147483647

Where?: In addition to the form shown above, TxBASIC allows an optional variable to be specified that will receive the error code number and the address of the token that failed. This value can be examined in the error handling routine to decide what action to take.

NOTE: Be aware that all information on previous GOSUBs is lost. In addition, TxBASIC resets the token parameter stack.

The token address for the error found is stored in the least significant two bytes of the variable, and the error number in the most significant two bytes. Use the divide and mod operators to separate out these parts.

The token address can be used to look into the Token List file to get an idea of where the error occurred.

Example: **Write this program in TxTools:** (indentation for clarity only)

```

ONERR TROUBLE,E           // jump to TROUBLE if error, error # in E
A = 2                     // initialize variable
FOR I = 0 TO 99           //execute loop up to 100 times
A = A * 2                 // make A larger, possible error source
NEXT I                    // loop back
TRYTHIS:
  B = TEMP(1000000)       // another possible error source
  STOP                    // won't get here, second err causes exit

TROUBLE:
  PRINT "Error #", E/65536," found";
  PRINT " at token address ", #1H, E % 65536, "H"
  IF E/65536=7 PRINT "Multiply out of range" : GOTO TRYTHIS
  IF E/65536=14 PRINT "TEMP argument out of range"
  STOP

```

The following will be displayed when you run it in TxTools:

```

Error #7 found at token address 28H
Multiply out of range
Error #14 found at token address 35H
TEMP argument out of range

```

Remarks: As with CBREAK , ONERR must be executed to be effective. For this reason, you should put ONERR near the beginning of the program.

The error handler may be changed any number of times by executing ONERR with different arguments.

Cautions: Watch out! Being too fancy with this command can cause a disaster. If the example were a subroutine, the error jump would wash out the return information, and its RETURN statement would cause an error.

See Also: Refer to [“Tokenizer Flags Sub-Menu Option Descriptions \(IBM PC\)”](#) on page 3-17 or [“TxBASIC Options Sub-Menu Descriptions \(Macintosh\)”](#) on page 3-37 for information on creating the Token List file and for the format of this file.

NOTE: Will be cleared if “Boot from ROM” command is used by TxTools or if RAM has been found to be corrupted.

OTEXT

send characters from datafile

Syntax: OTEXT ptrvar [, \terminator][;]
OTEXT <v>[, \<x>][;]

Description: The OTEXT command sends text from the logger's datafile. The specified variable 'ptrvar' is used as a pointer. A single line can then be off-loaded over the hardware UART. Starting from the location pointed to by 'ptrvar', all characters including the terminating character (carriage return by default) are sent. The pointer is updated to the position following the carriage return after the data is sent. A carriage return and line feed are sent if the terminating character is a CR. A semicolon at the end of the statement will suppress the transmission of the terminator.

Example: Write this program in TxTools:

```
X=0
ITEXT X
X=0
PRINT "the string “;:OTEXT X;:PRINT ” was received”
```

The following will be displayed when you run it in TxTools:

```
HELLO
the string 'HELLO' was received
```

Remarks: If an out-of-memory error occurs in the middle of an OTEXT command, the variable will be left with the value it had before the line was executed. OTEXT takes about (n+4) ms to send n characters.

Cautions: OTEXT is potentially hazardous since it will make the Tattletale send characters until it encounters a carriage return (or other terminator) or runs out of logger memory. If there is no carriage return (or other terminator) in the datafile, and the pointer starts at zero, the command could take over four minutes to execute! The OTEXT output is buffered. PERIOD or COUNT should not be started while there are still characters in the buffer (see TSTOUT in [“Assembly Language Subroutines”](#) on page 4-41).

PCLR

set I/O line low

Syntax: PCLR pin [,pin...]
PCLR <x1> [,<x2>...]

Description: PCLR first makes the specified lines outputs, and then clears these lines to a logic low (0 volts).

Examples: // Use two ways to clear I/O lines
FOR A=0 TO 5 : PCLR A : NEXT A
PCLR 6,7,8,9,10,11,12,13

Remarks: In TxBASIC, when multiple lines are specified in a single command, the lines are handled in three blocks, the lines in the block 0 - 7, then the lines in the block 8 - 15, and finally the lines in the block 16 and above.

NOTE: PCLR will run up to 17 times faster if you use constant arguments. But to do this, you must specify the model number with the MODEL command. For the example above you will have to use the MODEL command if all the arguments are constants. The MODEL command tells the tokenizer what Tattletale the program is for so the command can be optimized for the specific digital I/O pin configuration for that Tattletale model.

Cautions: Remember, a Model 6 in its HALT mode (and a Model 2B in pseudo-HALT mode), will have its I/O lines as inputs, only weakly asserted by pull-down resistors (I/O line 4 has a pull-up). The same is true for the Model 4A DONE command and the Model 5's DONE-like command.

See Also: PSET on [page 5-75](#), PTOG on [page 5-76](#), PIN on [page 5-71](#) and MODEL on [page 5-62](#). Refer to the **“Digital Output Protection”** procedure on page 7-15 in **Section 7 - Application Notes**.

PEEK

read memory byte

Syntax: PEEK (<addr>)
PEEKL(<addr>) Return four bytes at address *addr*.
PEEKW(<addr>) Return two bytes at address *addr*.

Description: This function returns the value of the byte(s) located at the address <addr> in parentheses. The <addr> can be an expression that evaluates to an address.

Examples: (indentation for clarity only)

```
print peek ($H74C0)
A= peek (&H112)
```

```
start_prog:
  print "square of values from 0 to 9"
  for a = 0 to 9
    b = a * a
    print #5, a, b
  next a
  print "last b value bytes: ";
  print #02H, peek(varptr(b)), " ", peek(varptr(b)+1), " ";
  print #02H, peek(varptr(b)+2), " ", peek(varptr(b)+3)
  print "program starts at ", #04H, labptr(start_prog)
  print "program ends at ", #04H, labptr(end_prog)
end_prog:
```

Remarks:

See Also: POKE on [page 5-72](#).

PERIOD

measure period of signal

Syntax: PERIOD (count, timeout)
PERIOD (<x1>,<x2>)

Description: PERIOD measures the amount of time it takes for 'count' cycles of a signal to pass. The input signal must be connected to I/O line 13 (the Model 8 uses I/O line 4) and is measured in units of 1/1.2288 μ Sec (about 0.81380 μ Sec) for a Tattletale with a 4.9152 MHz crystal, and 1/2.4576 μ Sec (about 0.40690 μ Sec) for a Tattletale using a 9.8304 MHz crystal. The units of measurement on the Model 8 are derived from the value returned by the TPUGetTCR1 () extension. The units are the reciprocal of this value. If 'timeout' * 0.01 seconds passes before the prescribed number of cycles transpires, the returned value will be zero. This keeps the Tattletale from locking up forever if no signal is at the input. Period will return incorrect values for input frequencies higher than 15 KHz for 4.9MHz Tattletaes or 30KHz for 9.8MHz Tattletaes (50 KHz for Model 8).

Examples: **Write this program in TxTools:** (indentation for clarity only)

```

                print "COUNT gives ",count(100)," Hz"
                X = PERIOD (100,100)
                if X=0 print "PERIOD gives 0 Hz": goto finish
Non-Model 8:   print "PERIOD gives ",122880000/X," Hz"
Model 8 only:  print "PERIOD gives ", (TPUGetTCR()*100)/X, "Hz"
finish:       stop

```

The following will be displayed when you run it in TxTools:

```

COUNT gives 4997 Hz
PERIOD gives 4996 Hz

```

Cautions: Be careful when dividing anything by PERIOD, since PERIOD can return a zero which would cause a 'HOW?' error.

The maximum value for the count and timeout arguments is 65535.

PERIOD may not work properly if there is activity on the serial port.

Remarks: Argument maximum is 65535.

See Also: COUNT on [page 5-26](#), TSTOUT in [“Assembly Language Subroutines”](#) on page 4-41 and Extension TPUGetTCR1 on [page 5-131](#).

PIN

read state of I/O line

Syntax: value = PIN (pin [,pin...])
value = PIN (<x1> [,<x2>...])

Description: For each line that is specified by the PIN instruction, the data direction for that line is set to input. A value is then returned that is formed from the states of the specified lines. If the voltage at a particular line is above 2.0 volts, the PIN instruction interprets the input as a 1; if it is below 0.7 volts, it is interpreted as a 0. Intermediate values will return indeterminate results.

This command returns a value of all listed lines in a set order, not depending on the order they are listed in the command.

I/O Line	Weight	I/O Line	Weight	I/O Line	Weight
0	128	9	16384	17	1048576
1	64	10	8192	18	2097152
2	32	11	1024	19	4194304
3	16	12	512	20	33554432
4	8	13	256	21	67108864
5	4	14	65536	22	134217728
6	2	15	131072	23	268435456
7	1	16	262144	24	1073741824
8	32768				

NOTE: For the Model 6F, I/O line 14 and I/O line 15 are reversed.

See Also: PCLR on [page 5-68](#), PSET on [page 5-75](#), PTOG on [page 5-76](#) and MODEL on [page 5-62](#). Refer to the **“Digital Input Protection”** procedure on page 7-16 in **Section 7 - Application Notes**. Refer to **Section 6 - Hardware Details** for model specific pin numbers.

Remarks: **NOTE:** PIN will run up to 17 times faster if you use constant arguments. But to do this, you must specify the model number with the MODEL command. You will have to use the MODEL command if all the arguments are constants. The MODEL command tells the tokenizer what Tattletale the program is for so the command can be optimized for the specific digital I/O pin configuration for that Tattletale model.

POKE

place byte into RAM

Syntax: POKE <addr>,<value>
POKEL<addr>,<value> Copy four bytes of *value* to four bytes at address *addr*.
POKEW<addr>,<value> Copy two bytes of *value* to two bytes at address *addr*.

Description: These commands store 1, 2 or 4 bytes of the value of the expression <value> at the address that results from the evaluation of the expression <addr>.

Examples: poke &H74C0,123

```
b=123456789 // write 123456789 to locations
for a=&H74C3 to &H74C0 step -1 // 112H -115H, msb first
poke a,b%256
b=b/256
next a
```

Remarks:

See Also: PEEK on [page 5-69](#) and VARPTR on [page 5-109](#).

For floating point numbers, two numbers can be used to separately specify the minimum width and the number of decimal places of precision. One or both must be present with the default digits of precision being 6.

Characters: Individual characters can be sent by preceding their ASCII value with a backslash. You can also specify a variable after the backslash. The least significant byte stored in the variable will be sent.

Examples:

```
print "Strike the bell, second mate!",\7, "Let us go below!"
// Next example prints letters 'a' through 'z' then CR/LF
for i = 97 to 122 : print \i; : next i : print
```

Datafile blocks: {<x1>,<x2>} specifies a block of the datafile starting at location <x1> and ending with location <x2>. These bytes are sent as single 8-bit characters.

Examples:

```
print {0,99}
```

Trailing Semicolon: A trailing semicolon causes TxBASIC to omit the CR, LF that is normally sent at the end of a print statement.

Remarks: To send a CR without a LF at the end of a line use \13 to send the carriage return, and a semicolon to suppress the normal CR LF.

Cautions: PRINT's output is buffered. The interrupts from transmitted character may interfere with PERIOD or COUNT (see TSTOUT in [“Assembly Language Subroutines”](#) on page 4-41).

See Also: There are many other examples of PRINT statements in this section. SDO, USEND, STORE and CALL can have similar formats.

PSET

set I/O line high

Syntax: PSET pin [,pin, pin,...]
PSET <x1> [,<x2>...]

Description: This command sets the data direction register for the specified lines to outputs, and then sets the lines to a logic high (+5 volts).

Examples: FOR A=0 TO 13: PSET A: NEXT A

PSET 0,1,2,3,4,5,6,7,8,9,10,11,12,13

Either one of these will set all of the output lines high (useful in an application where the I/O lines are not used, and you do not wish to add pull-ups or pull-downs to keep the inputs in a low-power state).

Remarks: In TxBASIC, when multiple lines are specified in a single command, the lines are handled in three blocks, the lines in the block 0 - 7, then the lines in the block 8 - 15, and finally the lines in the block 16 and above.

NOTE: PSET will run up to 17 times faster if you use constant arguments. But to do this, you must specify the model number with the MODEL command. For the example above you will have to use the MODEL command if all the arguments are constants. The MODEL command tells the tokenizer what Tattletale the program is for so the command can be optimized for the specific digital I/O pin configuration for that Tattletale model.

Cautions: During the period between the execution of the Model 6's HALT and its wakeup all I/O lines will become inputs. The same is true for the Model 2B's pseudo-HALT, and the Model 5's DONE-like command.

See Also: PCLR on [page 5-68](#), PTOG on [page 5-76](#), PIN on [page 5-71](#) and MODEL on [page 5-62](#). Refer to the **“Digital Output Protection”** procedure on page 7-15 in **Section 7 - Application Notes**.

PTOG

toggle I/O line

Syntax: PTOG pin [,pin, pin,...]
PTOG <x1> [,<x2>...]

Description: The PTOG command sets the data direction register for the specified lines to outputs, and then changes the lines to the opposite state they held before this command was executed.

Examples: FOR A=0 TO 13: PTOG 1: NEXT A

This example will cause I/O line 1 to change state 14 times ending up in its original state.

Remarks: In TxBASIC, when multiple lines are specified in a single command, the lines are handled in three blocks, the lines in the block 0 - 7, then the lines in the block 8 - 15, and finally the lines in the block 16 and above.

NOTE: PTOG will run up to 17 times faster if you use constant arguments. But to do this, you must specify the model number with the MODEL command. For the example above you will have to use the MODEL command if all the arguments are constants. The MODEL command tells the tokenizer what Tattletale the program is for so the command can be optimized for the specific digital I/O pin configuration for that Tattletale model.

Cautions: During the period between the execution of the Model 6's HALT and its wakeup all I/O lines will become inputs. The same is true for the Model 2B's pseudo-HALT, and the Model 5's DONE-like command.

See Also: PCLR on [page 5-68](#), PSET on [page 5-75](#), PIN on [page 5-71](#) and MODEL on [page 5-62](#). Refer to the **“Digital Output Protection”** procedure on page 7-15 in **Section 7 - Application Notes**.

QUIT enter 'end of program' low power mode

NOTE: This command is for the Models 6F and 8 only.

Syntax: QUIT

Description: This command puts the Tattletale into its lowest power mode where it has a typical power drain of about 100 μ A (running from a 10V supply). After executing QUIT, the Tattletale wakes every minute and checks the serial line for a break. If the serial line is in the spacing state (high) for example while receiving a break, the Tattletale will exit QUIT as though it had received a CTRL-C. The only other exit from QUIT is by a power-up reset.

In order to minimize its current drain, during QUIT the 6F reduces its operating voltage to 4.0 Volts. QUIT switches the Model 8 to 3.0 Volts; this behavior cannot be overridden with HybAt3V.

Example:

```

X=0
ONERR labelB :REM low power mode when datafile filled
SLEEP 0
labelA: BURST X,5,2
        SLEEP 100
        GOTO labelA
labelB:  QUIT
  
```

Remarks: Upon exiting QUIT mode the ? variable will be reset to 0 and the Tattletale will be running at RATE 1. All I/O lines will be restored to the state they had before QUIT was executed.

Cautions: While in the dormant mode (after executing 'QUIT'), all of the I/O lines are converted to inputs and are not asserted except by their pull-down resistors (I/O line 4 has a pull-up). Any hardware attached to the Tattletale should drop into its minimum power drain state when the lines are so asserted. In order to make use of the low power QUIT state you must provide terminations for several of the Tattletale's lines:

Line	Termination	Remarks
I/O lines	1M to +5 or gnd	I/O 4 already terminated

RATE

change SLEEP interval

Syntax: RATE <speedup>
RATE <x>

Description: This command allows, in some applications, faster operation of the Tattletale. To keep downward compatibility, the '?' variable incrementing interval is kept at 10mS. The new RATE command allows you to adjust the SLEEP interval to several new values, while the '?' variable is incremented at the same rate of 100Hz. Although the RATE command allows arguments to 255, only those rates that are factors of 192 (96 for 4.9152MHz) will give proper intervals (the Model 8 can go up to 200).

The following are valid for all Models except the Model 8:

Command	Sleeps per sec	Interval	'?' update rate
RATE 1	100	10mS	100 per sec
RATE 2	200	5mS	100 per sec
RATE 3	300	3.33mS	100 per sec
RATE 4	400	2.5mS	100 per sec
RATE 6	600	1.66mS	100 per sec
RATE 8	800	1.25mS	100 per sec
RATE 12	1200	833.33µSec	100 per sec
RATE 16	1600	625µSec	100 per sec
RATE 24	2400	416.6µSec	100 per sec
RATE 32	3200	312.5µSec	100 per sec
RATE 48	4800	208.33µSec	100 per sec
RATE 64*	6400	156.25µSec	100 per sec
RATE 96	9600	104.166µSec	100 per sec
RATE 192*	19200	52.0833µSec	100 per sec

* Only available with a 9.8304MHz clock frequency.

The following are valid only for the Model 8:

Command	Sleeps per sec	Interval	'?' update rate
RATE 1	100	10mS	100 per sec
RATE 2	200	5mS	100 per sec
RATE 4	400	2.5mS	100 per sec
RATE 5	500	2mS	100 per sec
RATE 8	800	1.25mS	100 per sec
RATE 10	1000	1mS	100 per sec
RATE 20	2000	500µSec	100 per sec
RATE 25	2500	400µSec	100 per sec
RATE 40	4000	250µSec	100 per sec
RATE 50	5000	200µSec	100 per sec
RATE 100	10000	100µSec	100 per sec
RATE 200	20000	50µSec	100 per sec

Examples: RATE and SLEEP can be combined to reach unusual measurement intervals; the interval below is 1/240 sec.

```
RATE 12
SLEEP 0
FOR A=1 TO 10
  SLEEP 5
  PRINT "X"
NEXT A
```

Remarks: ADLOOP extends the regular housekeeping interrupt (that occurs at every 'RATE' interval) from about 40µSec to about 260µSec. This gives a maximum practical RATE value of 32 for the Model 2B, 5F or 5F-LCD, and 24 for the Model 6 (since at least 50µSec uninterrupted must be left to the foreground for the Model 6's disk operations to not interfere with the ADLOOP measurements.

The default value of RATE (at power-up and after a reset) is 1.

Cautions: Values of <x> not shown in the table above will lead to unexpected results.

The UGET and USEND commands will be affected by Rate values other than 1.

Add the command SLEEP 0 immediately following the RATE command for the new rate to take effect.

REM

rest of line is a remark

Syntax: REM --- anything ---
// Anything
' Anything

Description: 'REM' is reserved for compatibility with TT BASIC programs. REM is equivalent to the normal Tx BASIC comments: double slashes (ignore rest of line) or an apostrophe (must be first character on line, the line is ignored).

Examples: REM Program 17 revision 3 -- LH 12-12-86
'
'First test datafile for independence
x=0 : rem initialize datafile pointer
yPoint=4 // two backslashes also cancel out the rest of the line

Cautions: In Tx BASIC, all program statements starting with REM are discarded before being loaded to the Tattletale. Be careful to put a space or tab after the REM command, though, or the Tx BASIC tokenizer will misinterpret it.

REPEAT

execute loop until expression true

Syntax: REPEAT
 ...commands...
 ...to be executed...
 UNTIL <x>

Description: REPEAT loops provide one of four methods of looping available in TxBASIC. The code between the REPEAT and UNTIL commands will be executed until 'expression' becomes true. Unlike the FOR and WHILE loops, the testing of 'expression' takes place after the loop has executed so a REPEAT loop will always run at least once. Because this structure stores nothing on the stack, these loops can nest as deeply as you like. GOTO will exit any number of nested REPEAT loops.

Examples: (indentation for clarity only)

```
// Example 1 - force input to be 0 or 1 and count mistakes
begin:  tries = 0
        print "Input 0 to exit or 1 to continue"
        repeat
            input "Continue? " goAgain
            tries = tries + 1
        until goAgain = 0 | goAgain = 1

        print "that took you ",tries," tries"

        iff goAgain = 0
            print "Program terminating"
            gosub CleanUp
            stop
        else
            goto begin
        endif

// Example 2 - force input to be between 0 and 100, quit on 99
// count number of mistakes and quit after 10
repeat
    tries = 0
    repeat
        input newNumber
        tries = tries + 1
        if tries >= 10 goto give_up
    until newNumber < 100 & newNumber > 0
until newNumber = 99
print "99 was input. Time to stop."
stop
give_up:
    print "Too many mistakes!"
    stop
```

Remarks: The REPEAT loop is only available with version 2.00 or later.

See Also: FOR on [page 5-40](#), GOTO on [page 5-45](#) and WHILE on [page 5-112](#).

RIGHT

return right most characters

Syntax: RIGHT(str\$, x)

Description: Return a string made up of the right-most *x* characters of *str\$*.

Examples: astring\$ = "Throw out everything but the last word"
print right(astring,4)

Remarks: This function can be used anywhere a string variable or string constant would be used.

See Also: LEFT on [page 5-56](#), MID on [page 5-61](#), INSTR on [page 5-51](#), and LEN on [page 5-57](#)

RTIME**move ? variable data to ? array**

Syntax: RTIME, RTIME <v>

NOTE: The 6F and 8 Models can not use the RTIME <v> option.

Description: RTIME translates the 5-byte '?' variable to the '?' array.

NOTE: For the Model 6F, the RTIME command transfers the information from the 6F's real time clock to the '?' array.

Time is counted in 1/100 ths of a second starting at New Years Day 1980. The 5th byte of the '?' variable is only accessible through RTIME and STIME. This command, with STIME, allows setting and reading the Tattletale's real-time clock.

? (0) gets the second (0 to 59)

? (3) gets the day (1 to 31)

? (1) gets the minute (0 to 59)

? (4) gets the month (1 to 12)

? (2) gets the hour (0 to 23)

? (5) gets the year (0 to 99).

Examples: **Write this program in TxTools:** (indentation for clarity only)

```

    sleep 0
loop: rtime
    print "THE TIME IS ",?(2),":",?(1),":",?(0);
    print " on ",?(4),"/",?(3),"/",?(5)
    sleep 100
    goto loop

```

The following will be displayed when you run it in TxTools:

THE TIME IS 16:24:18 on 5/8/86

THE TIME IS 16:24:19 on 5/8/86

THE TIME IS 16:24:20 on 5/8/86

THE TIME IS 16:24:21 on 5/8/86

...

RTIME <v> translates the variable <v> to the '?' array. Time is counted in seconds (hundredths of seconds not included) starting at New Years Day 1980. In both cases dates beyond 2040 are out of range. Leap years are handled properly. For the Model 8, years beyond 2038 are out of range.

Cautions: The ? array is already defined, do not initialize it with DIM!

If you copy the 5-byte ? variable to a 4-byte TxBASIC variable, you lose the most significant byte of the ? variable. Use RTIME <v>.

See Also: STIME on [page 5-95](#).

RUN start background program

NOTE: This command is not available for the Model 8.

Syntax: RUN <label>

Description: 'RUN' in TxBASIC launches a background task. The background program is rigorously timed, interrupting the foreground task as needed to perform its job. <label> defines the start of the background task. Background tasks are stopped by STOP or a reset. Only one background task can be enabled at a time. All commands are available to the background task that are available to the foreground task except ADLOOP.

When a background task is running, the SLEEP timing for the foreground is no longer rigorous (it's easy to oversleep). So the * print is disabled for the foreground task - ONLY WHILE A BACKGROUND TASK IS RUNNING. To distinguish between SLEEP commands, the background task prints a tilde '~' if it oversleeps instead of *.

All variables are available to both the foreground and background tasks.

Use the RUN command to launch a background task. Follow the RUN with the line number or label of the routine you want to execute in the background. For example "RUN back1" will cause the program starting at the label "back1" to run in the background. The program starting at label "back1" must have a SLEEP or STOP somewhere to pass control back to the foreground, or the foreground task will never have a chance to run!

When a background task is started with RUN, it doesn't start immediately. A flag is set in TxBASIC that shows it is available to start. It waits in suspended mode until the next clock interrupt occurs (every 10msec or the rate chosen with the RATE command) which truly starts the background task.

Cautions: Background and foreground operations can interact since the background task has priority. When running a background operation you should not run:

Command	Failure
USEND, UGET	Both use software timing which is interfered with by a background task interrupting them.
TONE, PERIOD COUNT	Timing can be incorrect or input edges can be missed.

Do not use RETURN to end the background as it will cause a stack failure, crashing the program.

Remarks: RUN and dual tasking are not available for the Model 8.

See Also: [“Dual Tasking”](#) on page 4-25. There is an example there, too.

SCROLL

Scrolling LCD output

NOTE: This command is for the 5F-LCD only.

Syntax:

```
SCROLL "s"[\<x>][, #c][, val][, {<x2>, <x3>}][;]
```

Description:

This command has the same syntax as print, but displays the data to the 5F-LCD's LCD. The data is scrolled onto and off of the display. This allows the display to show long messages. The characters that can be displayed are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, c, C, d, E, F, g, H, h, i, J, l, L, n, N, o, O, P, R, r, S, t, u, U, y, -, , ?, ., =, * (* is drawn as °).

```

ABCDEFGHIJ KLMNOPQRST UVWXYZ
AbCdEfGhIj L nOp rStU y
 c h i n o f u
? * = 1 2 3 4 5 6 7 8 9 0 -
? ° = 1 2 3 4 5 6 7 8 9 0 -

```

Figure 5-6: 5F-LCD Characters that can be Displayed

NOTE:

The display has only seven segments plus a decimal point, and accordingly some characters can not be displayed. The undisplayable characters will be skipped if an upper case or lower case equivalent cannot be substituted either.

See Also:

Disply on [page 5-32](#).

SDI

shift register input

NOTE: The Model 8 uses I/O line 7 for the SDI command (instead of I/O line 9).

Syntax: value = SDI (bits)
value = SDI (<x>)

Description: SDI is designed to bring in a serial data stream from a shift register and return with a value formed from this data stream. SDI first applies a negative going pulse to I/O line 3. This pulse is used to latch data into the shift registers. SDI then shifts in the number of bits of data specified by the value in parentheses, using I/O line 9 as the data input line and I/O line 5 as a clock. The returned value is made from the binary data received (msb first). Clocking occurs on the positive edge. This works nicely with a 74HC165 or 74HC166 shift register.

The command `LET A = SDI N` will cause N bits of data to be clocked in to form an N-bit two's complement number. If less than 32 bits are shifted in, the unspecified MSB's are zeros. The last bit shifted in has a weight of 1, with the preceding bits given weights of 2, 4, 8, 16, etc. If 32 bits are shifted in, the first bit will be the sign bit of the resulting two's complement number. Figure 5-7 shows 17 bits shifted in to form the number 134EDH, which is 79085 decimal.

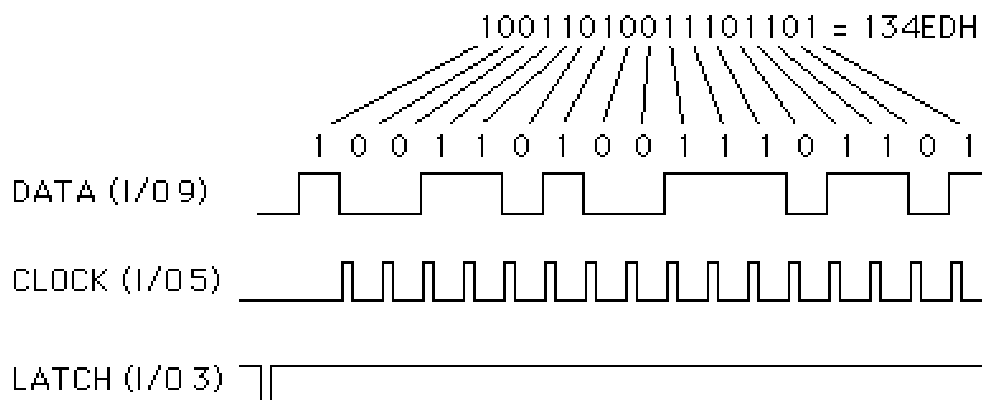


Figure 5-7: Example of 17 Bits being Shifted

This sequence works well with both 74HC165 and 74HC166 shift registers. Figure 5-8 uses two 74HC165s to shift in 16 inputs. By cascading two more shift registers, 32 inputs can be read at once.

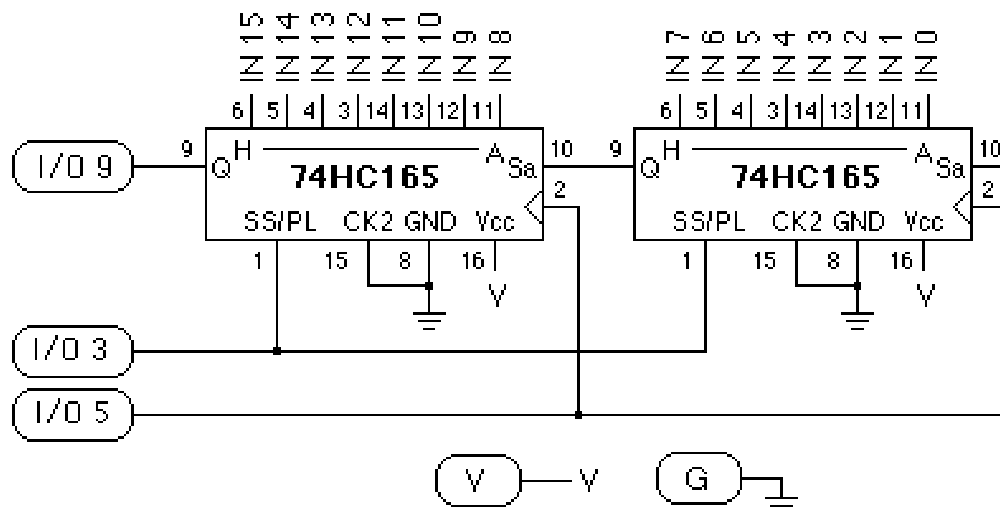


Figure 5-8: 74HC165 and 74HC166 Shift Registers

Remarks: I/O lines 3, 5 and 9 will normally be connected to a shift register such as a 74HC165 or 74HC166 as shown above.

Since the most significant bit of the shift register attached to I/O line 9 is always available, an initial 'clock' is not needed. Thus, if you request N bits, there will be N-1 'clock' pulses on I/O line 5.

SDI is not available for the Models 5F-LCD and 7.

Cautions: Run time errors will occur if the value in parentheses is not in the range 1 to 32.

See Also: SDO on [page 5-88](#).

SDO

shift register output

NOTE: The Model 8 uses I/O line 8 for the SDO command (instead of I/O line 11 as shown below).

Syntax: SDO value, bits
 SDO <x1>,<x2>
 or
 SDO "string"[\,ascii val][,#format][,val][,{df start, df end}]
 SDO "<s>"[\,<x>][,#c][,val][,{<x2>, <x3>}][;]

Description:

Form 1: SDO is designed to send a serial data stream made up of the least significant bits of the value. The bits are sent out I/O line 11, using the positive edge of I/O line 5 as a clock. After the last shift pulse, I/O line 2 is used as a positive-going latch pulse. This works nicely with a 74HC595 shift register.

Form 2: The second form of SDO sends characters out the serial line (eight bits and then latch). The form is the same as that used in PRINT, except that a string must be sent first (a zero-length string is fine). This form of SDO is particularly useful for driving some LCD displays. See "**Add a 16 x 2 Display to the Tattletale**" procedure in **Section 7 - Application Notes**.

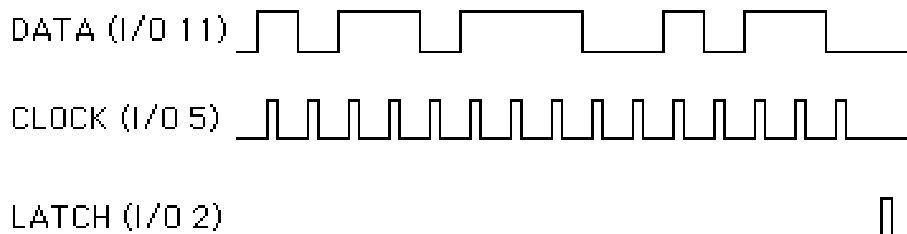


Figure 5-9: Timing Lines used by the SDO Command

Figure 5-9 shows the timing of the three lines used in the SDO command. The circuit in Figure 5-10 shows one use of SDO. Here two 74HC595 shift registers are cascaded to form 16 outputs. By adding two more 74HC595's, a total of 32 output lines can be changed with a single SDO command.

NOTE: These format specifiers, #D, #H, #B, #F, #S or #Q are considered ambiguous. Do D or H signify the radix (decimal or hexadecimal) or a variable field width? You can use variables for field width but not with those 12 names (upper or lower case). To get the radix form, use #1D, #1H etc.

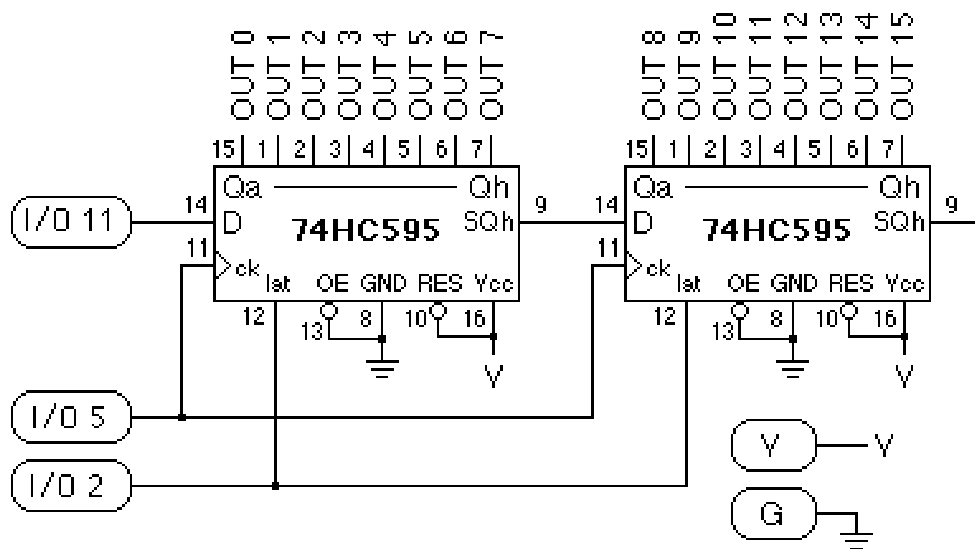


Figure 5-10: SDO Command Timing Lines

Remarks: I/O lines 2, 5 and 11 will normally be connected to a shift register such as a 74HC595 as shown in Figure 5-10.

Since the bit on I/O line 11 is always available to be stored in the least significant bit of the shift register, a final 'clock' is not needed. Thus, if you request N bits, there will be N-1 'clock' pulses on I/O line 5.

SDO is not available for the Models 5F-LCD and 7.

Cautions: In form 1, the bit's value must be in the range 1 to 32.

See also: SDI on [page 5-86](#) and PRINT on [page 5-73](#).

SIN

sine

Syntax: value = SIN(<x>)

Description: SIN returns the sine of the expression in parentheses. The argument must be in degrees. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
degrees! = 0.0           // init arg (notice '!' means it's a float)
result! = 0.0           // just to force 'result' to be a float
for i = 1 to 6
  result = sin(degrees)
  print "The sine of ",#5.1F,degrees," is ",#6.3F,result
  degrees = degrees + 72.0
next i
```

The following will be displayed when you run it in TxTools:

```
The sine of 0.0 is 0.000
The sine of 72.0 is 0.951
The sine of 144.0 is 0.588
The sine of 216.0 is -0.588
The sine of 288.0 is -0.951
The sine of 360.0 is 0.000
```

Remarks: Don't forget, the argument to this function is in degrees, not radians.

See Also: COS on [page 5-25](#), TAN on [page 5-100](#) and ATN on [page 5-17](#).

SLEEP

wait for an interval

Syntax: SLEEP tics
SLEEP <x>

Description: SLEEP places the Tattletale in a dormant mode until the number of 1/100-second (or other interval specified by the RATE command) intervals specified by the tics argument has expired. It does this by comparing the specified 15-bit tics value with the 16-bit value in a free-running counter, called the interval counter, that is incremented every 1/100 of a second. When a match is found, it clears the interval counter and completes the instruction. In this manner, the SLEEP command actually sets intervals between SLEEP commands, and accordingly, is independent of other timing delays. If the comparison shows that the interval counter has a larger value than the tics value specified by the command, the warning '*' will be printed. If 'tics' = 0, no check is made, but the interval counter is reset to zero.

Every background task must have a SLEEP if the foreground is to run concurrently with the background.

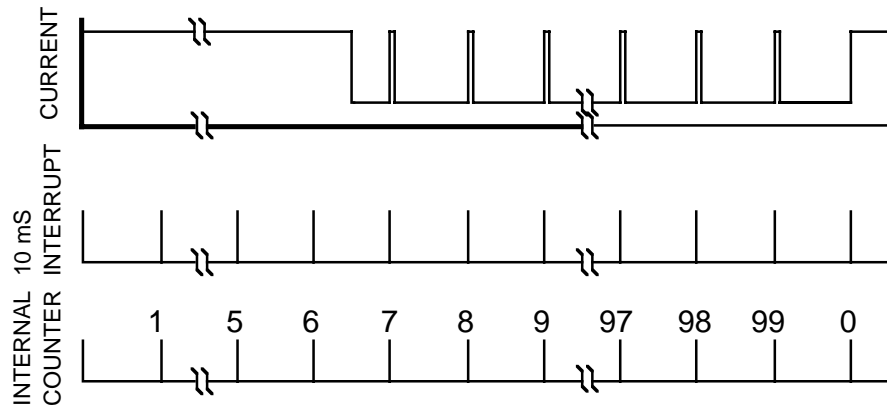


Figure 5-11: Timing Chart for a Sleep 100 Command

Figure 5-11 shows the timing for a SLEEP 100 command that was separated from the previous SLEEP command by commands that took about 65mS to execute (notice the current drain is high for about 6 1/2 tics). If the intervening commands had taken more than a second, the current drain would not have dropped, the interval counter would have been reset to zero and a '*' would have been sent out the hardware UART. In TxBASiC, if SLEEP 'oversleeps' when it is used in the background, a tilde '~' will be printed instead of an asterisk '*'. If a background task is running, the foreground '*' error will not be printed. A special memory location is changed if an oversleep occurs. Bit 0 at address B6 hex is set if the foreground oversleeps. Bit 1 is set if the background oversleeps.

NOTE: This memory location IS NOT CLEARED AT RESET! If you want to use this feature, clear this byte when you start your program with: POKE &HB6,0

SLEEP used with Optional Commands

If there are commands (not separated by a colon) on the same line with the SLEEP command, these commands will be executed if I/O line 1 goes high before the timeout (when the interval counter reaches the 'tics' value). If I/O line 1 does not go high before the timeout, these commands will not be executed.

If I/O line 1 going high causes an early exit from SLEEP, the *interval counter is not reset*. Thus, it is possible to retain synchronization by finishing off the interval with another SLEEP command. The purpose of the I/O line 1 exit is to permit immediate response to an external event. If the event servicing routine's execution time, added to the count already in the interval counter, is greater than the 'tics' value in the next SLEEP command, an '*' response will be returned from that SLEEP command which will in turn reset the interval counter, thereby losing the time synchronization. To eliminate this problem following an overrun, this form of the SLEEP command resets the interval counter to the actual time minus the intended wakeup time. This allows the SLEEP command to borrow time from the next SLEEP, ensuring that synchronization is not lost. To use this feature without using an I/O line 1 wakeup, follow the SLEEP with a REM. For this to work properly, SLEEP must be initialized with a SLEEP 0 command; otherwise, it will do a lot of borrowing before it gets synchronized!

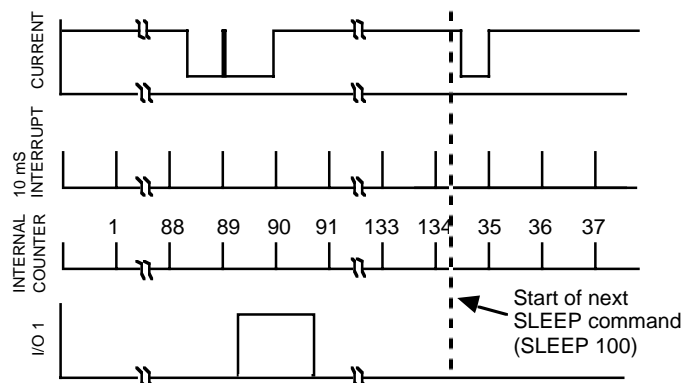


Figure 5-12: Timing Chart Showing Sleep Command Waking Early

Figure 5-12 shows the use of a SLEEP command awakened early by I/O line 1. By the time the next SLEEP command is reached, the interval counter value is greater than the SLEEP command's argument; the SLEEP exits almost immediately and reduces the interval counter by the amount of the argument.

I/O line 1 is level sensitive, so the interrupt routine must clear the interrupt before continuing (otherwise a single interrupt can awaken the Tattletale several times). I/O line 1 must be left as an input, or set to an input (X=PIN(1) will do it) before the SLEEP command.

```

REM *** SLEEP demonstration program ***
REM Sends a 10Hz signal until I/O line 1 goes high
Label1: SLEEP 0      :REM initialize sleep counter
        SLEEP 9 STOP :REM wait 0.09 sec
        PSET (0)
        SLEEP 1      :REM wait .01 sec
  
```

PCLR (0)
GOTO Label1

Remarks: The SLEEP command is central to the Tattletale logger, providing it with low power consumption and also a method for synchronizing its activities. Vital to its operation is an interval counter that is incremented every 1/100 of a second.

Since the sleep interval includes the time it takes to execute the instructions between SLEEP instructions, the intervals are very accurate. The SLEEP time should be long enough to allow execution of the intervening instructions.

Cautions: Intervals must have a value between zero and 32767. Numbers outside that range will cause an error message. Use SLEEP 0 to reset the interval counter.

Early exit from Sleep by an I/O line 1 interrupt will not occur until the first 10mS interrupt has expired, because TxBASiC does not check the I/O line state before going to Sleep.

Examples: SLEEP 0
Label1: SLEEP 1000 GOSUB label2: PSET(0): PCLR(0): GOTO Label1
GOTO Label1

Here I/O line 0 is used to clear the interrupt coming in on I/O line 1. This main loop executes a subroutine at label2 every time an interrupt comes along.

```
SLEEP 100 REM
```

This version of sleep can borrow time from the next SLEEP on a timing overrun, whereas

```
SLEEP 100 : REM
```

will not. What a difference a colon makes!

SQR

square root

Syntax: value = SQR(<x>)

Description: SQR returns the square root of the expression in parentheses. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
start:
  input "SQR("argument!";
  print ") = ",#1F,sqr(argument)
  goto start
```

The following will be displayed when you run it in TxTools:

```
SQR(1) = 1.000000
SQR(2) = 1.414214
SQR(4) = 2.000000
SQR(100) = 10.000000
SQR(1000) = 31.622778
SQR(1000000) = 1000.000000
SQR(123.4) = 11.108556
SQR(-1) = NaN "Not-a-Number" because sqr(-1) is imaginary
SQR(16) = 4.000000 Note that the NaN error didn't stop program
```

Remarks: Taking the square root of a negative number returns a value of Not-a-Number (specified by the IEEE floating point specification). This will not stop program execution. It sets the Not-a-Number bit in the floating point error variable FPERR.

STIME**set software real-time-clock**

Syntax: STIME, or STIME <v>

NOTE: The 6F and 8 Models can not use the STIME <v> option.

Description: STIME translates the '?' array to the 5-byte '?' variable.

NOTE: For the Model 6F, the STIME command transfers the information from the 6F's real time clock to the '?' variable.

Time is counted in 1/100ths of a second starting at New Years Day 1980. The 5th byte of the '?' variable is only accessible through RTIME and STIME. This command, with RTIME, allows setting and reading the Tattletale's real-time clock.

Examples: **Write this program in TxTools:**

```
input "THE YEAR IS (0 TO 99) "?(5)
input "THE MONTH IS (1 TO 12) "?(4)
input "THE DAY IS (1 TO 31) "?(3)
input "THE HOUR IS (0 TO 23) "?(2)
input "THE MINUTE IS (0 TO 59) "?(1)
input "THE SECOND IS (0 TO 59) "?(0)
stime
STOP
```

The following will be displayed when you run it in TxTools:

```
THE YEAR IS (0 TO 99) 86
THE MONTH IS (1 TO 12) 5
THE DAY IS (1 TO 31) 7
THE HOUR IS (0 TO 23) 15
THE MINUTE IS (0 TO 59) 23
THE SECOND IS (0 TO 59) 45
```

STIME <v> translates the '?' array to the variable <v>. Time is counted in *seconds* (hundredths of seconds are lost) starting at New Years Day 1980. In both cases dates beyond 2040 are out of range. For the Model 8, years beyond 2038 are out of range. Leap years are handled properly.

Cautions: The ? array is predefined. Do not initialize it with DIM!

If you copy the 5-byte ? variable to a 4-byte TxBASIC variable, you lose the most significant byte of the ? variable. Use RTIME <v>.

See Also: RTIME on [page 5-83](#).

STOP

stop program execution

Syntax: STOP

Description: This command stops TxBASIC execution. A STOP command is not needed at the end of the program. A background task must use a STOP to end its execution to pass control back to the foreground task.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
        gosub psub
        stop
psub:  print "A very dull sample program"
        return
```

The following will be displayed when you run it in TxTools:

A very dull sample program

Remarks: A program can have any number of STOP lines.

STORE

store to datafile

Syntax: STORE ptrvar [[,#size] [,expr] . . .]
 STORE <v> [[,#c] [,<x>] . . .]
 or
 STORE ptrvar, "string" [,#format][,val][,\ascii val][,{df start,df end}][;]
 STORE ptrvar, "<s>" [,#c][,<x1>][,\<x2>][,{<x3>,<x4>}][;]

TxBASIC determines which storage form is being used by checking if the first value is a string or not. If it is a string, even zero length (""), it is form 2, otherwise it will be interpreted as form 1.

Form 1

Description: STORE places bytes or words or double words into the datafile, starting at the location pointed to by the variable 'ptrvar'. Formats #1, #2, and #4 are used to specify whether the data is to be stored as one, two or four bytes. #1 specifies that the least significant 8 bits be stored, #2 specifies that the least significant 16 bits be stored, and #4 specifies that all 32 bits be stored. The pointer variable is updated after the data has been stored. The default format of store is in bytes so that the #1 can be omitted in byte storage commands.

If expr is a string variable, the string's length is stored as one byte and then each character of the string is stored as one byte per character. In this case, the #c specifier is ignored. When reading a binary string, use GETS().

Examples: Write this program in TxTools:

```
X=0 : REM store five temperature measurements
FOR A=1 TO 5
  T=TEMP(CHAN(7))
  PRINT #4,T/100,".",#02,T%100;           // print them too
  STORE X,#2,T
NEXT A: PRINT
X=0                                       // Recover them, and print them
FOR A=1 TO 5
  T=GET(X,#2):PRINT #4,T/100,".",#02,T%100;
NEXT A
```

The following will be displayed when you run it in TxTools:

```
21.14 21.14 21.14 21.14 21.14
21.14 21.14 21.14 21.14 21.14
```

Form 2

In this second form, STORE can place characters into the datafile with the same flexibility that one can send characters using the PRINT command. Any mix of strings, ASCII values, values and blocks of datafile can be stored as long as the first one is a string (a zero length string will do). See PRINT on [page 5-73](#) for more details.

```
X=0  
STORE X,"7",{0,99};
```

This short example fills the first 101 locations of the datafile with the ASCII character '7'. It copies a '7' directly to the first location, then copies location 0 to location 1, location 1 to 2 etc. This is just one example of block moves that can be made.

```
X=0:RTIME  
STORE X,"Peak of ",A,"at ",#02,?(2),"?",?(1),"?",?(0)
```

This example shows how you could build up a file in your Tattletale's datafile so that it could be off-loaded as text later.

The second form of STORE does not store the length byte of the string. It just stores each character of the string as one byte per character.

NOTE: These format specifiers, #D, #H, #B, #F, #S or #Q are considered ambiguous. Do D or H signify the radix (decimal or hexadecimal) or a variable field width? You can use variables for field width but not with those 12 names (upper or lower case). To get the radix form, use #1D, #1H etc.

Remarks: As with all data storage commands, the pointer variable is incremented after the completion of the command, leaving the variable pointing to the location immediately following the last location stored to.

To see the maximum datafile address for your Tattletale, look at the read-only constant "dfmax". For example, print DFMAX will print the maximum datafile address.

Cautions: If an out-of-memory error occurs in the middle of a STORE command the variable will be left with the value it had before the line was executed.

See Also: GET on [page 5-42](#), GETS on [page 5-43](#) and PRINT on [page 5-73](#).

STR **return a string representing the value of *expr*.**

Syntax: STR(expr)

Description: Return a string representing the value of *expr*. The type of the conversion (float or integer) is determined by the type of the expression (as decided by the tokenizer). Floating point numbers are converted into scientific notation with six decimal point precision.

Examples:
print str(5+6)
print str(1.103 - 2.028)

Remarks: This function can be used anywhere a string variable or string constant would be used.

See Also: VAL on [page 5-108](#)

TAN

tangent

Syntax: value = TAN(<x>)

Description: TAN returns the tangent of the expression in parentheses. The argument must be degrees. This function takes a floating point argument and returns a floating point result. If the argument is an integer, it will be converted to float first.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
degrees! = 0.0          // init arg (notice '!' means it's a float)
result! = 0.0          // just to force 'result' to be a float
for i = 0 to 6
  result = tan(degrees)
  print "The tangent of ", #5.1F, degrees, " is ", #6.3F, result
  degrees = degrees + 72.0
next i
```

The following will be displayed when you run it in TxTools:

```
The tangent of 0.0 is 0.000
The tangent of 72.0 is 3.078
The tangent of 144.0 is -0.726
The tangent of 216.0 is 0.726
The tangent of 288.0 is -3.078
The tangent of 360.0 is 0.000
```

Remarks: Don't forget, the argument to this function is in degrees, not radians.

See Also: COS on [page 5-25](#), SIN on [page 5-90](#) and ATN on [page 5-17](#).

TEMP

convert number to temperature

Syntax: value = TEMP(<x>)

Description: TEMP converts the 0–65535 reading 'x' that results from a measurement of one of the A to D channels to a temperature, assuming that the channel is connected to a voltage divider as shown in the diagram below. The temperature conversion is given in hundredths of degrees C. This will return the board temperature if 'x' = CHAN(7) for Model 2Bs as these have onboard temperature sensors, or if one has been added to the Model 5F, 5F-LCD, 6, 6F or 8. The circuit used for the onboard thermistor is shown in Figure 5-13. The capacitor is optional and can be useful if the thermistor is mounted on a long cable.

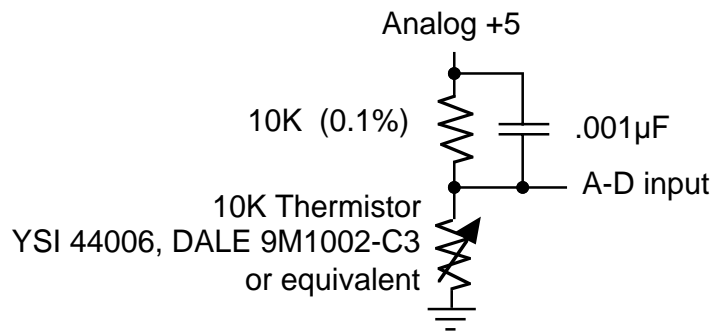


Figure 5-13: Thermistor Circuit

The "analog+5" connection is to Vsw for the Model 2B, and Vref (or Vref+) for all other models.

Examples: Write this program in TxTools:

```
for A = 0 to 65520 step 256
  tempValue! = temp(A)/100.
  print #7.2f, tempValue;
  if A / 256 % 10 = 9 print
next A
```

The following will be displayed when you run it in TxTools:

166.00	166.00	166.00	166.00	165.88	158.48	151.08	143.68	136.33	132.00
127.68	123.35	119.05	116.12	113.20	110.27	107.36	105.16	102.96	100.76
98.57	96.82	95.07	93.32	91.57	90.10	88.62	87.15	85.68	84.40
83.13	81.85	80.58	79.48	78.38	77.28	76.18	75.18	74.18	73.18
72.18	71.31	70.43	69.56	68.68	67.86	67.03	66.21	65.38	64.61
63.83	63.06	62.28	61.58	60.88	60.18	59.48	58.83	58.18	57.53
56.89	56.26	55.64	55.01	54.39	53.81	53.24	52.66	52.09	51.51
50.94	50.36	49.79	49.26	48.74	48.21	47.69	47.16	46.64	46.11
45.59	45.09	44.59	44.09	43.59	43.11	42.64	42.16	41.69	41.24
40.79	40.34	39.89	39.44	38.99	38.54	38.09	37.64	37.19	36.74
36.29	35.86	35.44	35.01	34.59	34.16	33.74	33.31	32.89	32.49
32.09	31.69	31.29	30.89	30.49	30.09	29.69	29.29	28.89	28.49
28.09	27.69	27.29	26.89	26.49	26.11	25.74	25.36	24.99	24.59
24.19	23.79	23.39	23.01	22.64	22.26	21.89	21.49	21.09	20.69
20.29	19.91	19.54	19.16	18.79	18.41	18.04	17.66	17.29	16.91
16.54	16.16	15.79	15.39	14.99	14.59	14.19	13.81	13.44	13.06
12.69	12.29	11.89	11.49	11.09	10.71	10.34	9.96	9.59	9.19
8.79	8.39	7.99	7.59	7.19	6.79	6.39	5.96	5.54	5.11
4.69	4.26	3.84	3.41	2.99	2.56	2.14	1.71	1.29	0.86
0.44	0.01	0.41	0.89	-1.36	-1.84	-2.31	-2.79	-3.26	-3.74
-4.21	-4.69	-5.16	-5.64	-6.11	-6.64	-7.16	-7.69	-8.21	-8.76
-9.31	-9.86	-10.41	-10.99	-11.56	-12.14	-12.71	-13.34	-13.96	-14.59
-15.22	-15.87	-16.52	-17.17	-17.82	-18.57	-19.32	-20.07	-20.82	-21.62
-22.42	-23.22	-24.02	-24.97	-25.92	-26.87	-27.82	-28.90	-29.97	-31.05
-32.13	-33.48	-34.83	-36.18	-37.53	-39.36	-41.18	-43.01	-44.85	-47.65
-50.45	-53.25	-56.00	-56.00	-56.00	-56.00				

#

A printout that showed all of the different temperatures that can be represented by a Tattletale's 12-bit converter would be sixteen times as long as the listing above.

Remarks:

No attempt has been made to resolve temperatures greater than 166° C or less than -56° . The conversion gives about 0.03° resolution for temperatures between about 0 and 35°C.

TONE

send square wave out

Syntax: TONE period, count
TONE <x1>,<x2>

Description: TONE allows you to produce a square wave of predetermined period and length at I/O line 12 (the Model 8 uses I/O line 6). The expression 'period' gives the square wave period, which is measured in multiples of 1.6276 μ Sec for Tattletalets with a 4.9 MHz crystal. For Tattletalets with a 9.8MHz crystal, period is in multiples of 0.8138 μ Sec (multiples of 1.0 μ Sec for Model 8). The range of period is 60 (about 10240Hz or 20480Hz for fast crystals) to 65535 (about 9.5Hz or 19Hz for fast crystals). The range for the Model 8 is 40 (25 KHz) to 65535 (15.25 Hz). The period argument has a different resolution for the Model 8 than the other Tattletalets and it varies with the system frequency. To get the resolution (in seconds), in floating point, use the TPUGetTCR1() extension in this manner: Resolution! = 1.0 / TPUGetTCR1(). The expression 'count' gives the number of cycles for the second argument, which can be as few as one, or as many as 32767 cycles. The Model 8 cannot count cycles of the TONE. Instead, the 'count' argument is really a time-out value in (0.01 second ticks). Therefore, if you use a 'count' argument of 1000, the TONE will be produced for 10 seconds. This output can be used to drive a speaker and produce notes covering a large portion of the audio spectrum. TONE sets I/O line 12 to an output and leaves I/O line 12 low at the end of a command (the Model 8 does this to I/O line 6).

A special form of TONE is available to produce a continuous square wave. If you use a value of 0 for the count argument, the square wave will continue (at the period you select) until you execute a TONE with a zero value for both the period and count arguments. You can change the period of the continuous signal by using TONE with a new period argument and a zero count argument. The square wave period will change on the next transition of the signal. This form of TONE is not available on the Model 8.

Example 1:

```
dim @(25)
X=1000000/4/44/16           // Two octaves above 440 'A'
for A=24 to 0 step -1
  @(A)=X
  X=X*106/100
next A                       // Have half tone scale from @(0) to @(23)
Y=500000
tone @(21),Y/@(21)
tone @(23),Y/@(23)
tone @(19),Y/@(19)
tone @(5),Y/@(5)
tone @(14),Y/@(14)         // Recognize that?
```

Example 2:

```
sleep 0
for i = 10 to 1 step -1
  tone i*60, 0              // update continuous signal period
  sleep 10                  // produce this frequency for 100 mSec
next i
tone 0, 0                   // stop the tone
```

Remarks: In the example above, the number of cycles is normalized by the interval to make the length of the note constant. The above poorly-tempered scale is actually remarkably close to a well-tempered scale. `PRINT 10000*@(24)/@(0)` yields the value 2515, about 0.7% away from the right number (2500). The 1.06 ratio is good to about 1/10 of a half tone per octave.

Cautions: Driving circuitry should expect the quiescent state to be low. If the TONE command is not used for a while after power-up, set that line low with a PCLR 12 command (or PCLR 6 for the Model 8) to ensure that the speaker driver is in its low power state!

Watch out: TONE 65000,32000 will growl for about two hours on a Tattletale with a 4.9MHz crystal!

UGET bring character in software UART

NOTE: The Model 8 uses I/O line 14 for the UGET command (instead of I/O line 7).

Syntax: UGET baud, count, ptrvar, timeout
UGET <x1>, <x2>, <v>, <x3>

Description: The UGET command treats I/O line 7 as a UART input line to receive serial data. The input expects CMOS levels and has the marking state high, inverted from the normal RS-232 sense. The baud rate is specified by the expression 'baud' and can be any value between 110 baud and 4800 (4.9152MHz crystal) or 9600 (9.8304MHz crystal). On the Model 8, the baud rate can be as high as 38400. The expression 'count' specifies the number of bytes to be received. 'Ptrvar' specifies the pointer variable to be used to store the data in the datafile as it is received. The data is stored directly into memory and is not echoed to the screen. The expression 'timeout' specifies the time-out (in 1/100ths of a second); this time-out aborts the receiving routine in case of an external system failure and starts at the beginning of the execution of the command. time-out can be set to any value between 0.01 and 655.35 seconds.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```
      LET X=0
      UGET 4800,10,X,800
      IF X<10 GOTO LT10
      PRINT "RECEIVED ALL":STOP
LT10:  PRINT "RECEIVED ONLY ",X," CHARACTERS"
```

The following will be displayed when you run it in TxTools:

RECEIVED ONLY 0 CHARACTERS

This program will try to receive 10 characters at 4800 baud, storing them starting at the current location of the X variable, and timing out if all ten are not received in eight seconds. Provisions are also made to detect and deal with external system failure.

Remarks: UGET uses I/O line 7 as a UART input line. Like USEND, this line uses CMOS levels and has a marking high state, inverted from the normal RS-232 sense. The transfer format for the received characters is eight data bits, no parity, with one stop bit (at least). Remember that the variable (ptrvar) is incremented with each byte received. The 'count' parameter has no maximum.

Cautions: Whereas USEND works at 9600 baud, UGET will not work reliably at rates above 4800 baud for Tattletales with 4.9152MHz crystals. 9600 baud will work for those models with 9.8304MHz crystals. Since the starting level is sensed, not the edge, holding the line low will appear as a continuous stream of zeroes except for the Model 8.

USEND send characters out software UART

NOTE: The Model 8 uses I/O line 13 for the USEND command (instead of I/O line 8).

Syntax: USEND baud, count, ptrvar
 USEND <x1>, <x2>, <v>
 or
 USEND baud, "string" [,#format] [,var] [,\ascii val][,{df start, end}] [;]
 USEND <x1>, "<s>" [[,#c] [,<x2>] [,\<x3>][,{df start,df end}][;]

Description: USEND sends serial data out I/O line 8. It has two forms. One sends 'count' number of bytes from the datafile starting at the location pointed to by 'ptrvar' at the baud rate specified by 'baud' (110 to 9600 baud or 110 to 38400 for the Model 8). 'Ptrvar' is updated after the instruction has been completed. The second form of USEND has the identical syntax as PRINT, except that the baud rate must be specified by 'baud', and the first thing sent must be a string (a zero-length string is acceptable). Double quotes must be used as string delimiters. See the PRINT command on [page 5-73](#) for additional information.

Example 1 **Write this program in TxTools:** (indentation for clarity only)

```
start: X=0:ITEXT X
      Y=0:USEND 300,X,Y
      USEND 300,"",\10;
      GOTO start
```

The following will be displayed when you run it in TxTools:

THIS IS A TEST

This program echoes the keyboard input to the auxiliary serial port. The line that reads "USEND 300,"",\10;" sends the line feed stripped out by ITEXT.

Example 2

```
      USEND 9600,"      seconds from power-up";
loop: X=?
      USEND 9600,"",\13,#8,X/100,".",#02,X%100;
      GOTO loop
```

This simple program continuously displays the time on a terminal connected to I/O line 8 through a level shifter.

Remarks: **All models except the model 8:** USEND uses I/O line 8 as a UART output line. Like UGET, this line uses CMOS levels and has a marking high state, inverted from the normal RS-232 sense.

All models: The transfer format for the transmitted characters is eight data bits, no parity, with one stop bit (at least). Remember that the variable (ptrvar) is incremented at each byte sent.

Model 5F-LCD Only: I/O pin 8, on the Tattletale Model 5F-LCD, is one of the keypad lines and can be difficult to use for general purposes. The USEND data line

can be shifted to I/O pin 6. To get this behavior, set bit 4 of the Configuration byte at the beginning of your program. This is needed EACH TIME YOU RUN YOUR PROGRAM because on power-up, bit 4 of the Configuration byte is cleared to zero. See the “Configuration Byte Information” section of the hardware section of this manual.

Cautions: There is no breakout from USEND, and sending 220K bytes at 110 baud will take about 6 hours!

VAL return the numeric value of *str\$*

Syntax: VAL(str\$)

Description: Return the numeric value of str\$. Since the string could represent an integer or a float (determined at run-time), there is no way for the tokenizer to know the type of the return value at compile-time. The use of VAL() is limited to variable assignments because the type of the variable can determine the type of the conversion. For instance:

this is legal: afloat! = val(astring\$)

but this is not: print val(astring\$)

Examples: astring = val("12345")
print astring
bstring! = val("123.45")
print bstring

Remarks: This function returns an integer or floating point value, not a string

See Also: STR on [page 5-99](#)

VARPTR

get address of named variable

NOTE: This command is not available for the Model 8.

Syntax: VARPTR (<v>)

Description: This function returns the address of the variable named in parentheses. Since variables are defined when they are first used in a program, the variable can be used here before being defined elsewhere. Also, the variable can be an array but it cannot include the index value. The value returned on the Model 7 will be meaningless because that Tattletale is not set up with a fixed memory map.

Be aware that VARPTR returns the address of the most significant byte of the variable. The least significant byte = VARPTR (data) + 3.

Examples: (indentation for clarity only)

```
print varptr(data) //address of beginning of 'data' array (MSB)
print varptr(data)+3 //address of beginning of 'data' array (LSB)
```

```
start_prog:
  print "square of values from 0 to 9"
  for a = 0 to 9
    b = a * a
    print #5, a, b
  next a
  print "last b value bytes: ";
  print #02H, peek(varptr(b)), " ", peek(varptr(b)+1), " ";
  print #02H, peek(varptr(b)+2), " ", peek(varptr(b)+3)
  print "program starts at ", #04H, labptr(start_prog)
  print "program ends at ", #04H, labptr(end_prog)
end_prog:
```

Remarks: This function will not work on the Model 8. The MODEL command must be used with this command to tell the tokenizer what Tattletale the program is for.

See Also: LABPTR on [page 5-55](#) and MODEL on [page 5-62](#).

VGET

get variable from EEPROM

Syntax: value = VGET (<x>)

Description: VGET returns the corresponding value stored in EEPROM.

Only locations 0 through 31 can be specified. There is no limit to VGETs. The command VSTORE puts unique values into the EEPROM.

Examples: **Write this program in TxTools:**

```
fltVal! = 1.2345
vstore 0,fltVal
print #H,vget(0)
print #6.4F,asflt(vget(0))
```

The following will be displayed when you run it in TxTools:

```
3F9E0419
1.2345
```

Remarks: VGET can be used to load sensor calibration parameters stored using VSTORE during instrument calibration. This permits instruments to have identical TxBASIC programs, while allowing compensation for variations in sensors etc.

All values stored in the EEPROM are assumed to be integers. If you saved a floating point value, use the ASFLT function to make sure it's interpreted as a float. See the example above.

The Model 4A and 5 has no EEPROM and VGET will not work on these Tattletales.

Cautions: When shipped, Models 2B and 6 have their serial number stored in EEPROM location 31. Model 6, 6-1M and 6F uses locations 29 and 30 to store configuration information.

See Also: VSTORE on [page 5-111](#) and ASFLT on [page 5-14](#).

VSTORE

store variable to EEPROM

Syntax: VSTORE <address>,<x>

Description: Store value of <x> at EEPROM <address>. Valid addresses are 0 to 31.

Examples: **Write this program in TxTools:** (indentation for clarity only)

```

fltValue! = 1.0           //initialize our variable
for I = 0 to 20           //first fill EEPROM with n*n
  vstore I,fltValue*fltValue //store it (note - no ASFLT)
  fltValue = fltValue + 1.0 //update the variable
next I
for I=0 to 20             //recover @(0) to @(20)
  print #6.1F,sqr(asflt(vget(I))); //print square root(note ASFLT)
  if I%10 = 9 print      //print CR every ten items
next I

```

The following will be displayed when you run it in TxTools:

```

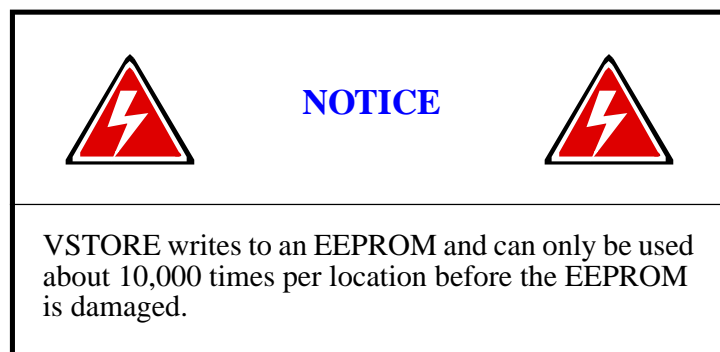
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0
20.0

```

Remarks: The Model 4A and 5 has no EEPROM. VSTORE will not work on these Tattletales.

Use the ASFLT function to recover floating point data from the EEPROM when you use VGET. You do not need to use ASFLT when you store the data.

Cautions: In the Model 6, 6-1M and 6F, locations 29, 30 and 31 are reserved for configuration information, writing to 29 or 30 will make the disk not work properly!



See Also: VGET on [page 5-110](#) and ASFLT on [page 5-14](#).

WHILE

execute loop while expression true

Syntax: WHILE <x>
...commands...
...to be executed...
WEND

Description: WHILE loops provide one of four methods of looping available in TxBASIC. The code between the WHILE and WEND commands will be executed as long as 'expression' is true. Like the FOR loop, and unlike the REPEAT loop, the testing of 'expression' takes place before the loop is executed. Because this structure stores nothing on the stack, you can nest these loops as deeply as you like. A GOTO can be used to exit any number of nested WHILE loops.

Example: (indentation for clarity only)

```
// collect data as long as I/O line 0 is low
onerr HandleError, errVar
dfPointer = 0
sleep 0
while pin(0) = 0
    store dfPointer, #2, chan(0), chan(2)
    sleep 100
wend
print "Finished logging, ready to offload"
stop

HandleError:
    iff errVar/65536 = 4
        print "Ran out of datafile. Stopped logging"
    else
        print "Logging stopped by error #",errVar/65536
    stop
```

Remarks: The WHILE loop is only available with version 2.00 or later.

See Also: FOR on [page 5-40](#), GOTO on [page 5-45](#) and REPEAT on [page 5-81](#).

XMIT+, XMIT-**enable, disable console output**

Syntax: XMIT-
XMIT+

Description: On receiving the XMIT- command, the Tattletale will stop echoing characters received by the main UART. XMIT- remains in effect until the Tattletale receives XMIT+ or a power-on reset. Under the influence of XMIT-, characters sent to the hardware UART are simply thrown away.

Examples: **Write this program in TxTools:**

```
xmit-  
print "You won't see this"  
xmit+  
print "You WILL see this"  
print "Input a number: ";  
xmit-  
input "this prompt will be lost" aNumber  
xmit+  
print "the number is ", aNumber
```

The following will be displayed when you run it in TxTools:

```
you WILL see this  
Input a number: <type 123 here> the number is 123
```

Remarks: XMIT- is most useful when you don't want characters entered into the Tattletale to be echoed.

Cautions: Remember, everything is suppressed by XMIT-, even error messages!

See Also: ITEXT on [page 5-53](#) for another method of not echoing input characters.

XSHAKE**enable X-on, X-off handshake**

NOTE: This command is not available for the Model 8.

Syntax: XSHAKE <x>
XSHAKE timeout

Description: XSHAKE enables CTRL-S, CTRL-Q handshake for the main UART. The expression <x> gives the time-out to re-enable transmission if no CTRL-Q is received (in 1/100ths of a second). Any value up to 655.35 seconds can be specified. XSHAKE 0 specifies no handshake, and is the default value at power-up. The rest of the time-out will be cancelled if a CTRL-C is received (and has not been disabled by clearing CENAB). The CTRL-C character is left in the serial buffer to be read by your program.

Examples: Write this program in TxTools:

```
SLEEP 0
XSHAKE 1000 : REM first fill @(n) with n*n
alph: PRINT "ABCDEFGHJKLMNOPQRSTUVWXYZ"
SLEEP 30
GOTO alph
```

The following will be displayed when you run it in TxTools:

```

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
(receives CTRL-S, stops transmitting until CTRL-Q received or 10 sec passes)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Remarks: The time-out will keep your program from hanging up even if the external controller locks up. XSHAKE remains in effect until explicitly altered.

XSHAKE is not available for the Model 8.

Cautions: The XON/XOFF handshake passes some control of your program timing to an external device, introducing delays not anticipated by your program, and potentially making it impossible to retain your program's synchronization.

XSHAKE must not be enabled during XMODEM transfers (OFFLD, ONLOAD, REMIND+ or REMIND)

Section 5 - TxBASIC Command Reference Addendum

NOTE: This addendum is for Model 8 users only. Each of these commands are considered to be extensions.

How to Use this Addendum

This section of the manual is to be used for reference for all the TxBASIC commands. The commands are listed alphabetically. The structure of each command generally follows this format:

TxBASIC Command Quick Reference

NOTE: The command names in Table 5-6 ARE case sensitive.

Table 5-6: TxBASIC Command Quick Reference

Command	Description
ADoff ()	Turns A-D converter power on.
ADon ()	Turns A-D converter power off.
Bipolar (chan)	Reads the A-D channel <i>chan</i> in bipolar mode.
Burst2KSetup (...)	Set up the fast A-D burst routine.
BurstAD (mode)	Enable or disable the fast A-D burst routine.
BurstInfo (mode)	Get status of fast A-D burst routine.
ConvertVarPtr (var)	Returns the address of the variable requested.
CtrlCHandle (mode)	Enable or disable handling of Ctrl-C character.
CtrlCReset ()	Clear the count of Ctrl-C characters.
GetTickRate ()	Returns the current clock tick rate in ticks per second used by the sleep functions.
HPSleep ()	Primarily a delay mode where the model 8 stays in a running condition and the TPU serial ports can still gather characters.
HybAt3V (mode)	Enable or disable going into 3 Volt mode in HYB.
HybCheckForBREAK (mode)	Enable or disable checking for BREAK in HYB.
KbChar ()	Returns the next available character from the primary serial port.
KbFlush ()	Removes any characters received by the primary serial port from its input queue.
KbHit ()	Checks to see if a character has been received by the primary serial port (ie. a key pressed).
LPMode ()	Selects one of nine low power settings.
Reset ()	Resets the Tattletale or optionally resets to the TOM-8 monitor.
SerActivate ()	Force on the serial port driver
SerGetBaud ()	Depending on what arguments are passed, this function will return one of the following: 1) the current baud rate. 2) the closest baud rate for a given system clock frequency. 3) the baud rate divisor for a given baud rate and system clock frequency.
SerSetBaud ()	Returns the actual baud rate set by the arguments passed to it.
SerShutDown ()	Remove power from serial port driver.
SimGetFSys ()	Returns the actual system clock frequency in Hertz.
SimSetFSys ()	Sets the system clock as close as possible to the freq argument variable.
StopWatchStart ()	Starts the microsecond stopwatch counter and initializes it to zero for subsequent calls by StopWatchTime.

Table 5-6: TxBASIC Command Quick Reference (Continued)

Command	Description
StopWatchTime ()	Returns the current value in the microsecond stop watch counter.
TPUAccumPeriod ()	Returns the accumulated counts of the TCR1 clock for the number of input cycles selected by argument count from the selected input channel.
TPUAccumPulseWidth ()	Returns the accumulated high time pulse widths in counts of the TCR1 clock for the number of input cycles selected by argument count from the selected input channel.
TPUChangePWM ()	Changes the PWM values for an initialized TPU channel.
TPUCountChan ()	Changes the TPU pin used for the TxBASIC COUNT command to the pin <i>chan</i> .
TPUGetTCR1 ()	Returns the value of the TPU's Timer#1 frequency in Hz.
TPUoff ()	Remove power from Time Processor Unit
TPUon ()	Apply power to Time Processor Unit
TPUPeriodChan ()	Change the TPU pin used for the TxBASIC PERIOD command to the pin <i>chan</i> .
TPUSdiClkChan ()	Change the TPU pin used for the TxBASIC SDI command <u>clock</u> to the pin <i>chan</i> .
TPUSdiDatChan ()	Change the TPU pin used for the TxBASIC SDI command <u>data</u> to the pin <i>chan</i> .
TPUSdiLchChan ()	Change the TPU pin used for the TxBASIC SDI command <u>latch</u> to the pin <i>chan</i> .
TPUSdoClkChan ()	Change the TPU pin used for the TxBASIC SDO command <u>clock</u> to the pin <i>chan</i> .
TPUSdoDatChan ()	Change the TPU pin used for the TxBASIC SDO command <u>data</u> to the pin <i>chan</i> .
TPUSdoLchChan ()	Change the TPU pin used for the TxBASIC SDO command <u>latch</u> to the pin <i>chan</i> .
TPUSetupPWM ()	Sets up a TPU channel as a pulse width modulated output channel.
TPUToneChan ()	Change the TPU pin used for the TxBASIC Tone command to the pin <i>chan</i> .
TPUUGetChan ()	Change the TPU pin used for the TxBASIC UGET command to the pin <i>chan</i> .
TPUUsendChan ()	Change the TPU pin used for the TxBASIC USEND command to the pin <i>chan</i> .
Trace ()	Turns on the trace function of TxBASIC.
TSerByteAvail ()	Checks to see if any bytes are available in the buffer of the specified channel.
TSerClose ()	Closes the TPU serial port specified as chan.
TSerGetByte ()	Gets the next byte from the channel input queue or waits for the next byte if no bytes are available.
TSerInFlush ()	Flushes the channels input queue and discards any characters in this queue.
TSerOpen ()	Opens a TPU channel for serial I/O.
TSerPutByte ()	Writes a byte of data to the channels output queue.
TSerResetBaud ()	Attempts to change the current baud rate for the selected channel to the new baud rate specified by the argument baud.

Explanations:

long:	signed 4 byte value	+2,147,483,647
ulong:	unsigned 4 byte value	+4,294,967,294
int or short:	signed 2 byte value	+32,768
ushort:	unsigned 2 byte value	+65535

Name: **ADon()**

Description: Turn A-D converter power on - checks to see if power was really off and adds a delay for reference to stabilize if power was off.

Arguments: none

Example: extension ADon,ADoff
 ADon()
 data = chan(0)
 ADoff()

Returns No return value.

See Also: ADOFF () on [page 5-117](#)

Name: **ADoff()**

Description: Turn A-D converter power off and sets a flag so ADon can tell power was really off. This saves about 1.5 mA of current drain. Any command that uses the A-D converter (like BURST and CHAN) will attempt to turn on the A-D converter if it has been turned off by this extension.

Arguments: none

Returns: No return value.

Example: extension ADon,ADoff
 ADon()
 data = chan(0)
 ADoff()

See Also: ADon() on [page 5-116](#)

Name: **BiPolar(chan)**

Description: Reads the A-D channel *chan* in bipolar mode. Turns A-D power on if necessary. Does not turn A-D power off.

Arguments: **short chan** - channel number

Returns: **long** - the bipolar reading as a 16-bit number with the sign in the MS bit.

Example: extension BiPolar, ADon, ADoff
 ADon()
 data = BiPolar(0)
 ADoff()

See Also: CHAN command on [page 5-117](#)

Name: **Burst2KSetup(offset, size, rate, wrap, nchan, chans...)**

Description: Sets up the fast A-D burst routine. Make sure you list as many channels as you stated in 'nchan'. You can repeat a channel if you like. The channels DO NOT have to be in numerical order. Each channel MUST be a number from 1 to 8.

Arguments: **long** (offset) -

Where in datafile to start storing the data. Normally 0.

long (size) -

Number of bytes of storage area for data.

long (rate) -

Number of 500 microsecond ticks between groups of samples. To collect a set of readings at a 100 Hz rate, use 20 for 'rate'.

long (wrap) -

Flag specifying whether to wrap around when the end of the datafile is reached. If 'wrap' = 0, then this routine will stop when it reaches the end of the datafile otherwise, 'wrap' must be -1, the routine will wrap around to the start of the datafile when it reaches the end.

long (nchan) -

Number of A-D readings to make at each sample time. This value must be from 1 to 8.

long (chans) -

List of channels (separated with commas) to sample in the order they should be sampled.

Returns: Nothing

Example:

```
extension Burst2KSetup, BurstAD, BurstInfo
// 10000 byte buffer (1250 sets of data), 100 Hz, wrap
Burst2KSetup(0,10000,20,-1,4,8,1,7,2)
sleep 0
print BurstInfo(0) // is BurstAD running?
BurstAD(1) // start sampling
print BurstInfo(0) // is BurstAD running?
sleep 2500 // sample for 25 seconds (about two times around buffer)
BurstAD(0) // stop sampling
print "Burst wrapped ", BurstInfo(2), " times"
print "Data starts at ", BurstInfo(1)
```

See Also: BurstAD, BurstInfo

Name: **BurstAD(mode)**

Description: Enables or disables the fast A-D burst routine. If the command is set up for wrap mode (using Burst2KSetup()), this is the only way to stop sampling. To start sampling, use mode = 1. To stop sampling, use mode = 0. You can start and stop sampling at will and the routine will pick up where it left off.

Arguments: **long** -

mode = 1 to start sampling, = 0 to stop sampling

Returns: Nothing

Example: See Burst2KSetup

See Also: Burst2KSetup, BurstInfo

Name: **BurstInfo(mode)**

Description: Returns information about the currently running BurstAD routine.

Arguments: **long** -
mode = 0, return 1 if burst is sampling, or 0 if not.
= 1, return the start of the datafile (for wrap mode).
= 2, return number of times buffer has wrapped (for wrap mode).

Returns: See explanation in Arguments.

Example: See Burst2KSetup

See Also: Burst2KSetup, BurstAD

Name: **ConvertVarPtr(variable)**

Description: This function returns the address of the variable requested

Arguments: alphanumeric -
the variable's name

Returns: **long** -
returns the variables address

Example:
extension ConvertVarPtr
a = 100
print"the value of a is ",a," the address of a is ",ConvertVarPtr(a)

See Also:

Name: **CtrlCHandle(mode)**

Description: This command enables or disables handling of the Ctrl-C character (03 hex). If the argument 'mode' is non-zero, Ctrl-C will be handled (this is the power-on default). If the argument 'mode' is zero, the Ctrl-C character will be ignored until this extension is called with a non-zero argument or until the program exits. Be aware that Ctrl-C characters will continue to be detected even when handling is disabled. The Ctrl-C detect count should be reset with extension CtrlCReset() before re-enabling Ctrl-C handling.

Arguments: **long** -
mode = 0, Ctrl-C characters are ignored
= non-zero, Ctrl-C characters are handled normally

Returns: Nothing

Example:

```
extension CtrlCHandle, CtrlCReset
CtrlCHandle(0)    // ignore Ctrl-C
< other programming code >
CtrlCReset()     // flush out any Ctrl-Cs that arrived during program
CtrlCHandle(1)   // now respond to any later Ctrl-C characters
```

See Also: CtrlCReset

Name: **CtrlCReset()**

Description: This clears the Ctrl-C detect count. This function should be used before CtrlCHandle() is used to re-enable Ctrl-C handling unless you want to react to any Ctrl-C detected while handling was disabled.

Arguments: None

Returns: Nothing

Example: See CtrlCHandle

See Also: CtrlCHandle

Name: **GetTickRate()**

Description: This function returns the current clock tick rate (granularity) in ticks per second used by the sleep functions. The startup rate is 200 and can be increased with the RATE command.

Arguments: none

Returns: **long** - the current clock tick rate in ticks per second. This value is modified by the RATE command.

Example:

```
extension GetTickRate
print " tickrate = ",GetTickRate()
```

See Also: SLEEP command on [page 5-91](#), HPSleep extension on [page 5-120](#), RATE on [page 5-78](#)

Name: **HPSleep(ticks)**

Description: This function is used primarily for timing. All the model 8 subsystems remain powered and the TPU serial ports can still gather characters. TxBasic's SLEEP command allows the main UART to continue working.

Arguments: **int** -
0 to 32767 the number of ticks, as set by the rate command, to wait before exiting.

Returns: **int** -
 0 if ok
 1 if out of range

Example:
 extension HPSleep
 HPSleep(0): HPSleep(200)

See Also: GetTickRate extension on [page 5-120](#), SLEEP command on [page 5-91](#), HYB command on [page 5-47](#)

Name: **HybAt3V(mode)**

Description: Normally, the HYB command switches the Tattletale into 3 volt mode to conserve power until the HYB exits, where the Tattletale switches back to 5 volt mode. This new extension allows you to change this behavior. If the argument in HybAt3V() is 0, HYB will no longer switch the Tattletale to 3 volts. If the argument is any other number, it enables the 3 volt mode during HYB. The power-up default is for the 3 volt mode to be enabled. This extension does not affect the behavior of the QUIT command.

Arguments: **long** -
 mode = 0, HYB does not switch to 3 Volt mode
 = non-zero, HYB will switch to 3 Volt mode

Returns: Nothing

Example:
 extension HybAt3V
 hyb(0) // zero the HYB counter
 HybAt3V(0) // HYB will not use 3V mode
 hyb(30) // HYB at slightly higher power consumption
 HybAt3V(1) // HYB will use 3V mode
 hyb(30) // HYB at lower power consumption

See Also: HYB command

Name: **HybCheckForBREAK(mode)**

Description: Normally, the HYB command awakes every 10 seconds (if you HYB for at least that long!) to check for a BREAK condition on the main UART line. If it detects the BREAK, it exits the HYB early as if a Ctrl-C character were received. HybCheckForBREAK allows you to change this behavior. If the argument is 0, it stops this 10-second check. If the argument is any other number, it enables the 10-second check.

Arguments: **long** -
 mode = 0, HYB does not check for BREAK
 = non-zero, HYB will check for BREAK every 10 seconds

Returns: Nothing

Example:

```
extension HybCheckForBREAK
hyb(0) // zero the HYB counter
HybCheckForBREAK(0) // HYB can only exit on timing out
hyb (30) // you'll have to wait 30 seconds
HybCheckForBREAK(1) // HYB can break-out early
hyb (30) // apply a BREAK on RS-232 for >10 sec
```

See Also: HYB command

Name: **KbChar()**

Description: This function returns the next available character from the primary serial port.

Arguments: none

Returns: **int** -
if = -1
no character available
if > -1
character that was received

Example:

```
extension KbChar
inchar = KbChar()
```

See Also: KbHit on [page 5-122](#), KbFlush on [page 5-122](#)

Name: **KbFlush()**

Description: This function empties the primary serial port's input queue. All characters are discarded.

Arguments: none

Returns: **int** - Zero

Example:

```
extension KbFlush
KbFlush()// Flush keyboard buffer
```

See Also: KbHit on [page 5-122](#), KbChar on [page 5-122](#)

Name: **KbHit()**

Description: This function checks to see if a character has been received by the primary serial port (ie. a key pressed). It does not remove the character.

Arguments: none

Returns: **int** -
 if = 0 no byte available from the primary serial port(1).
 if > 0 a byte is available from the primary serial port(1).

Example:
 extension KbHit,KbChar
 if KbHit() > 0 inchar = KbChar()

See Also: KbFlush on [page 5-122](#), KbChar on [page 5-122](#)

Name: **LPMode(mode)**

Description: This function selects one of nine low power settings. Each power setting has a different clock frequency and operating current. The higher the frequency, the higher the operating current.

Arguments: int -
 The values for mode are from 1 to 9 with 9 being the highest current/frequency and 1 being the lowest current/frequency. Upon powerup, the model 8 defaults to mode 2.

mode	frequency	comment
1	640Khz	If baud <= 19200
1	1.28Mhz	If baud = 38400
1	1.92Mhz	If baud = 57600
2	3.68Mhz	
3	6.08Mhz	
4	7.36Mhz	
5	9.12Mhz	
6	11.20Mhz	
7	12.80Mhz	
8	14.72Mhz	
9	16.00Mhz	

Returns: **int** -
 return the previous mode selected

Example:
 extension LPMode
 print "the previous mode was ", LPMode(2)
 print "the present mode is ", LPMode(2)

See Also: SimSetFSys on [page 5-126](#), SerSetBaud on [page 5-125](#), SerGetBaud on [page 5-124](#)

Name: Reset(mode)

Description: If any argument is present, it resets to the TOM8 monitor. If no argument is present, it restarts TxBASIC (if a program is burned into the EEPROM, the program will be launched automatically after restarting TxBASIC).

Arguments: **int mode** -
(#) reset to the TOM8 monitor
() Restart TxBASIC (and jump to program if present)

Returns: none

Example:
extension Reset
input "0 to Exit " ans
if ans = 0 Reset(1)

Name: SerActivate()

Description: Force on the serial port driver. You won't normally need to use this because the driver has an automatic restart when it detects a character at its input.

Arguments: None

Returns: Nothing

Example:
extension SerActivate
SerActivate() // the main serial port is now active

See Also: SerShutDown

Name: SerGetBaud(baud, freq)[For the Primary serial port only.]

Description: Depending on what arguments are passed (as described below), this function will return one of the following:
1) the current baud rate.
2) the closest baud rate for a given system clock frequency.
3) the baud rate divisor for a given baud rate and system clock frequency.

Arguments: **long** -
1) baud rate or zero
2) system clock frequency or zero

Returns: **long** -
If baud rate = 0 and system clock = 0:
returns the current serial baud rate

If baud rate = 0 and system clock > 0:
returns the baud rate using the current baud rate divisor and the system clock frequency argument value

If baud rate > 0 and system clock = 0:
returns the baud rate divisor for the current system clock frequency and the baud rate argument value

If baud rate > 0 and system clock > 0:
returns the baud rate divisor for the baud rate argument value and the system clock frequency argument value.

Example:

```
extension SerGetBaud
print"current baud = ", SerGetBaud(0,0)
print"baud rate for 16Mhz = ", SerGetBaud(0,16000000)
print"divisor for 19200baud = ", SerGetBaud(19200)
print"divisor for 8Mhz and 19200baud = ", SerGetBaud(19200,8000000)
```

See Also: SerSetBaud on [page 5-125](#), LPMMode on [page 5-123](#)

Name: **SerShutDown()**

Description: Turn off the serial port driver. This saves a couple of milliamps of current. The driver has an automatic restart when it detects a character at its input so you won't normally need to use SerActivate().

Arguments: None

Returns: Nothing

Example:

```
extension SerShutDown
SerShutDown() // The main serial port is now powered off
```

See Also: SerActivate

Name: **SerSetBaud(baud, freq)** [For the Primary serial port only.]

Description: This function returns the actual baud rate set by the arguments passed to it as described below.

Arguments: **long** -
1) baud rate
2) system clock frequency or zero

Returns: **long** -
If freq = 0:
returns the actual baud rate set, using the current system clock value

If freq > 0:
returns the actual baud rate set, using the new system clock argument value

Example:

```
extension SerSetBaud
a(2) = SerSetBaud(9600,8000000)
a(5) = SerSetBaud(38400)
```

See Also: SerGetBaud on [page 5-124](#), SimSetFSys on [page 5-126](#), SerSetBaud on [page 5-125](#)

Name: **SimGetFSys()**

Description: This function returns the actual system clock frequency in Hertz

Arguments: None

Returns: **long** -
returns the present system clock frequency

Example:

```
extension SimGetFSys
print " System clock = ", SimGetFSys()
```

See Also: SimSetFSys on [page 5-126](#)

Name: **SimSetFSys(freq)**

Description: This function sets the system clock as close as possible to the freq argument variable. Valid frequencies are listed in Table 5-7.

Arguments: **long** -
new system frequency in Hertz

Returns: **long** -
the actual frequency set by the freq argument variable

Example:

```
extension SimSetFSys, SerSetBaud
a = SimSetFSys(1000000)
SerSetBaud(9600)
```

See Also: LPMode on [page 5-123](#), SerSetBaud on [page 5-125](#)

Table 5-7: 68332 Clock Rates for the 40000 Crystal

System Frequency	/---- SYNCR ----\ w0x0 w0x1 w1x0 w1x1	/-- CS Access nS -\ wt Owt lwt 2wt	/- Baud Error % -\ 9600 19.2 38.4 57.6	TMCR uS
0.160 MHz	0000	>1mS >1mS >1mS >1mS	4.0	25.000
0.320 MHz	0100 4000	>1mS >1mS >1mS >1mS	4.0	12.500
0.480 MHz	0200	>1mS >1mS >1mS >1mS		8.3333
0.640 MHz	0300 4100 8000	>1mS >1mS >1mS >1mS	4.0 4.0	6.2500
0.800 MHz	0400	>1mS >1mS >1mS >1mS		5.0000
0.960 MHz	0500 4200	>1mS >1mS >1mS >1mS	4.0	4.1667
1.120 MHz	0600	858 >1mS >1mS >1mS		3.5714
1.280 MHz	0700 4300 8100 C000	746 >1mS >1mS >1mS	4.0 4.0 4.0	3.1250
1.440 MHz	0800	659 >1mS >1mS >1mS		2.7778
1.600 MHz	0900 4400	590 >1mS >1mS >1mS	4.0	2.5000
1.760 MHz	0A00	533 >1mS >1mS >1mS		2.2727
1.920 MHz	0B00 4500 8200	486 >1mS >1mS >1mS	4.0 4.0	2.0833
2.080 MHz	0C00	446 927 >1mS >1mS		1.9231
2.240 MHz	0D00 4600	411 858 >1mS >1mS	4.0	1.7857
2.400 MHz	0E00	382 798 >1mS >1mS		1.6667
2.560 MHz	0F00 4700 8300 C100	356 746 >1mS >1mS	4.0 4.0 4.0	1.5625
2.720 MHz	1000	333 700 >1mS >1mS	9.6 9.6 9.6	1.4706
2.880 MHz	1100 4800	312 659 >1mS >1mS	4.0	1.3889
3.040 MHz	1200	294 623 952 >1mS	1.1 1.1	1.3158
3.200 MHz	1300 4900 8400	278 590 902 >1mS	4.0 4.0	1.2500
3.360 MHz	1400	263 560 858 >1mS	0.6 8.6	1.1905
3.520 MHz	1500 4A00	249 533 817 >1mS	4.0	1.1364
3.680 MHz	1600	237 508 780 >1mS	0.2 0.2 0.2 0.2	1.0870
3.840 MHz	1700 4B00 8500 C200	225 486 746 >1mS	4.0 4.0 4.0 4.0	1.0417
4.000 MHz	1800	215 465 715 965	0.2 7.8 7.8 7.8	1.0000
4.160 MHz	1900 4C00	205 446 686 927	4.0	0.9615
4.320 MHz	1A00	196 428 659 891	0.4 0.4	0.9259
4.480 MHz	1B00 4D00 8600	188 411 635 858	4.0 4.0	0.8929
4.640 MHz	1C00	181 396 612 827	0.7 7.3	0.8621
4.800 MHz	1D00 4E00	173 382 590 798	4.0	0.8333
4.960 MHz	1E00	167 368 570 771	0.9 0.9 0.9	0.8065
5.120 MHz	1F00 4F00 8700 C300	160 356 551 746	4.0 4.0 4.0	0.7812
5.280 MHz	2000	154 344 533 723	1.1 6.9 6.9	0.7576
5.440 MHz	2100 5000	149 333 516 700	4.0 9.6 9.6	0.7353
5.600 MHz	2200	144 322 501 679	1.3 1.3	0.7143
5.760 MHz	2300 5100 8800	139 312 486 659	1.3 4.0	0.6944
5.920 MHz	2400	134 303 472 641	1.4 6.6	0.6757
6.080 MHz	2500 5200	129 294 458 623	1.1 1.1 1.1 9.1	0.6579
6.240 MHz	2600	125 286 446 606	1.5 1.5 1.5	0.6410
6.400 MHz	2700 5300 8900 C400	121 278 434 590	0.8 4.0 4.0	0.6250
6.560 MHz	2800	117 270 422 575	1.6 6.3 6.3	0.6098
6.720 MHz	2900 5400	114 263 411 560	0.6 0.6 8.6	0.5952
6.880 MHz	2A00	110 256 401 546	1.8 1.8	0.5814
7.040 MHz	2B00 5500 8A00	107 249 391 533	0.4 4.0	0.5682
7.200 MHz	2C00	104 243 382 521	1.9 6.1	0.5556
7.360 MHz	2D00 5600	101 237 373 508	0.2 0.2 0.2 0.2	0.5435
7.520 MHz	2E00	98 231 364 497	2.0 2.0 2.0 2.0	0.5319
7.680 MHz	2F00 5700 8B00 C500	95 225 356 486	0.0 4.0 4.0 4.0	0.5208
7.840 MHz	3000	93 220 348 475	2.0 6.0 6.0 6.0	0.5102
8.000 MHz	3100 5800	90 215 340 465	0.2 0.2 7.8 7.8	0.5000
8.160 MHz	3200	88 210 333 455	2.1 2.1 9.6 9.6	0.4902
8.320 MHz	3300 5900 8C00	85 205 326 446	0.3 4.0	0.4808
8.480 MHz	3400	83 201 319 437	1.4 1.4 1.4	0.4717
8.640 MHz	3500 5A00	81 196 312 428	0.4 0.4 0.4	0.4630
8.800 MHz	3600	79 192 306 420	1.2 2.3 2.3	0.4545
8.960 MHz	3700 5B00 8D00 C600	77 188 300 411	0.6 4.0 4.0	0.4464
9.120 MHz	3800	75 184 294 404	1.1 1.1 5.7 1.1	0.4386
9.280 MHz	3900 5C00	73 181 288 396	0.7 0.7 7.3 0.7	0.4310
9.440 MHz	3A00	71 177 283 389	0.9 2.4 8.9 2.4	0.4237
9.600 MHz	3B00 5D00 8E00	69 173 278 382	0.8 4.0	0.4167
9.760 MHz	3C00	67 170 272 375	0.7 0.7 0.7 5.6	0.4098
9.920 MHz	3D00 5E00	66 167 267 368	0.9 0.9 0.9 7.1	0.4032
10.080 MHz	3E00	64 163 263 362	0.6 2.5 2.5 8.6	0.3968
10.240 MHz	3F00 5F00 8F00 C700	63 160 258 356	1.0 4.0 4.0	0.3906
10.560 MHz	---- 6000	60 154 249 344	1.1 1.1 6.9	0.3788
10.880 MHz	---- 6100 9000	57 149 241 333	1.2 4.0 9.6	0.3676
11.200 MHz	---- 6200	54 144 233 322	1.5 1.3 1.3 1.3	0.3571
11.520 MHz	---- 6300 9100 C800	52 139 225 312	1.3 1.3 4.0 4.0	0.3472

Table 5-7: 68332 Clock Rates for the 40000 Crystal (Continued)

System Frequency	/---- SYNCR ----\ w0x0 w0x1 w1x0 w1x1				/-- CS Access nS -\ wt 0wt 1wt 2wt				/- Baud Error % -\ 9600 19.2 38.4 57.6				TMCR uS
11.840 MHz	----	6400	----	----	49	134	218	303	1.2	1.4	6.6	6.6	0.3378
12.160 MHz	----	6500	9200	----	47	129	212	294	1.1	1.1	1.1	9.1	0.3289
12.480 MHz	----	6600	----	----	45	125	205	286	0.9	1.5	1.5	----	0.3205
12.800 MHz	----	6700	9300	C900	43	121	199	278	0.8	0.8	4.0	0.8	0.3125
13.120 MHz	----	6800	----	----	41	117	194	270	0.7	1.7	6.3	1.7	0.3049
13.440 MHz	----	6900	9400	----	39	114	188	263	0.6	0.6	0.6	4.0	0.2976
13.760 MHz	----	6A00	----	----	38	110	183	256	0.5	1.8	1.8	6.2	0.2907
14.080 MHz	----	6B00	9500	CA00	36	107	178	249	0.4	0.4	4.0	8.4	0.2841
14.400 MHz	----	6C00	----	----	34	104	173	243	0.3	1.9	6.1	----	0.2778
14.720 MHz	----	6D00	9600	----	33	101	169	237	0.2	0.2	0.2	0.2	0.2717
15.040 MHz	----	6E00	----	----	31	98	164	231	0.1	2.0	2.0	2.0	0.2660
15.360 MHz	----	6F00	9700	CB00	30	95	160	225	0.0	0.0	4.0	4.0	0.2604
15.680 MHz	----	7000	----	----	29	93	156	220	0.1	2.0	6.0	6.0	0.2551
16.000 MHz	----	7100	9800	----	28	90	152	215	0.2	0.2	0.2	7.8	0.2500
16.320 MHz	----	7200	----	----	26	88	149	210	0.2	2.1	2.1	9.6	0.2451
16.640 MHz	----	7300	9900	CC00	25	85	145	205	1.5	0.3	4.0	0.3	0.2404
16.960 MHz	----	7400	----	----	24	83	142	201	1.4	1.4	1.4	2.2	0.2358
17.280 MHz	----	7500	9A00	----	23	81	139	196	1.3	0.4	0.4	4.0	0.2315
17.600 MHz	----	7600	----	----	22	79	135	192	1.2	1.2	2.3	5.7	0.2273
17.920 MHz	----	7700	9B00	CD00	21	77	132	188	1.1	0.6	4.0	7.4	0.2232
18.240 MHz	----	7800	----	----	20	75	129	184	1.1	1.1	1.1	1.1	0.2193
18.560 MHz	----	7900	9C00	----	19	73	127	181	1.0	0.7	0.7	0.7	0.2155
18.880 MHz	----	7A00	----	----	18	71	124	177	0.9	0.9	2.4	2.4	0.2119
19.200 MHz	----	7B00	9D00	CE00	17	69	121	173	0.8	0.8	4.0	4.0	0.2083
19.520 MHz	----	7C00	----	----	16	67	119	170	0.7	0.7	0.7	5.6	0.2049
19.840 MHz	----	7D00	9E00	----	15	66	116	167	0.7	0.9	0.9	7.1	0.2016
20.160 MHz	----	7E00	----	----	15	64	114	163	0.6	0.6	2.5	0.6	0.1984
20.480 MHz	----	7F00	9F00	CF00	14	63	111	160	0.5	1.0	4.0	1.0	0.1953
21.120 MHz	----	----	A000	----	12	60	107	154	0.4	1.1	1.1	4.0	0.1894
21.760 MHz	----	----	A100	D000	11	57	103	149	0.2	1.2	4.0	6.8	0.1838
22.400 MHz	----	----	A200	----	10	54	99	144	1.5	1.5	1.3	1.3	0.1786
23.040 MHz	----	----	A300	D100	8	52	95	139	1.3	1.3	1.3	4.0	0.1736
23.680 MHz	----	----	A400	----	7	49	92	134	1.2	1.2	1.4	1.2	0.1689
24.320 MHz	----	----	A500	D200	6	47	88	129	1.1	1.1	1.1	1.5	0.1645
24.960 MHz	----	----	A600	----	5	45	85	125	0.9	0.9	1.5	4.0	0.1603
25.600 MHz	----	----	A700	D300	4	43	82	121	0.8	0.8	0.8	0.8	0.1562
26.240 MHz	----	----	A800	----	3	41	79	117	0.7	0.7	1.7	1.7	0.1524
26.880 MHz	----	----	A900	D400	2	39	77	114	0.6	0.6	0.6	4.0	0.1488
27.520 MHz	----	----	AA00	----	1	38	74	110	1.6	0.5	1.8	0.5	0.1453
28.160 MHz	----	----	AB00	D500	1	36	72	107	1.5	0.4	0.4	1.8	0.1420
28.800 MHz	----	----	AC00	----	0	34	69	104	1.3	0.3	1.9	4.0	0.1389
29.440 MHz	----	----	AD00	D600	-1	33	67	101	1.2	0.2	0.2	0.2	0.1359
30.080 MHz	----	----	AE00	----	-2	31	65	98	1.1	0.1	2.0	2.0	0.1330
30.720 MHz	----	----	AF00	D700	-2	30	63	95	1.0	0.0	0.0	4.0	0.1302
31.360 MHz	----	----	B000	----	-3	29	61	93	0.9	0.1	2.0	0.1	0.1276
32.000 MHz	----	----	B100	D800	-4	28	59	90	0.8	0.2	0.2	2.1	0.1250

Name: StopWatchStart()

Description: This function starts the microsecond stopwatch counter and initializes it to zero for subsequent calls by StopWatchTime.

Arguments: none

Returns: none

Example:

```
extension StopWatchStart
StopWatchStart() // start the stopwatch at zero
```

See Also: StopWatchTime on [page 5-129](#)

Name: StopWatchTime()

Description: This functions returns the current value in the microsecond stopwatch counter. This timer has a resolution of 25 microseconds with a 1 millisecond overhead at 16Mhz.

Arguments: none

Returns: **long** -
current value of the microsecond stopwatch counter in microseconds

Example:
extension StopWatchTime
print "StopWatchTime = ", StopWatchTime()

See Also: StopWatchStart on [page 5-128](#)

Name: Trace(traceflag)

Description: This function selectively enables the various trace functions of TxBASIC.

Arguments: **int** -
Acceptable parameter values are:
0 = Trace off (trace nothing)
1 = Trace line numbers only
2 = Trace line numbers and token numbers

To trace any or all of the following, (in addition to line and token numbers) add any or all the following constants to 2:

+4 to trace token names
+8 to trace instruction pointer IP
+16 to trace Stack Pointer SP
+32 to trace Stack Dump (5 longs)

Returns: none

Example: To trace line numbers, token numbers, token names and stack dump, sum the following values (2+4+32) and use the result (38) as the parameter to trace.

Trace (38)

See Also:

Name: TPUAccumPeriod(chan, count, priority, timeout)

Description: This function returns the accumulated counts of the TCR1 clock over the specified number of input cycles (selected by argument **count**) input to the selected TPU channel.

Arguments: **int** -

The following arguments are required:

- 1) TPU channel number
 - 2) Desired number of input cycles counted (1 to 255)
 - 3) The interrupt service priority. 1 is low, 2 is medium, and 3 is high
- the next argument is optional:

ushort value -

- 4) The timeout count is 10 times the tick rate set by the 'rate' command. This is the same rate as the sleep commands use.

Returns: **ulong** -

if > 0

accumulated counts of the period in TCR1 clock cycles.

if = 0

Timeout value exceeded

Example:

```
extension TPUAccumPeriod
Pper = TPUAccumPeriod(7, Pcount, 3, 200)
```

See Also: PERIOD command on [page 5-70](#), TPUAccumPulseWidth on [page 5-130](#), TPUGetTCR1 on [page 5-131](#)

Name: **TPUAccumPulseWidth(chan, count, priority, timeout)**

Description: This function returns the accumulated high time pulse widths in counts of the TCR1 clock for the number of input cycles selected by argument count from the selected input channel.

Arguments: **int** -

The following arguments are required:

- 1) TPU channel number
 - 2) Desired number of input cycles counted (1 to 255)
 - 3) The interrupt service priority. 1 is low, 2 is medium, and 3 is high
- the next argument is optional:

ushort value -

- 4) The timeout count is 10 times the tick rate set by the 'rate' command. This is the same rate as the sleep commands use.

Returns: **ulong** -

if > 0

accumulated counts of the high time pulse width in TCR1 clock cycles.

if = 0

Timeout value exceeded.

Example:

```
extension TPUAccumPulseWidth
Ptime = TPUAccumPulseWidth(7, Pcount, 3, 200)
```

See Also: TPUAccumPeriod on [page 5-129](#), TPUGetTCR1 on [page 5-131](#)

Name: **TPUChangePWM(chan, pwmhi, pwmper)**

Description: This function will change the PWM values for an initialized TPU channel.

Arguments: **int** -
1) TPU channel number
2) High time in TCR1 cycles [call TPUGetTCR1() for current value.
3) Period time in TCR1 cycles.

Returns: none

Example:
extension TPUChangePWM
TPUChangePWM(8, Ptime, Pper)

See Also: TPUSetupPWM on [page 5-135](#), TPUGetTCR1 on [page 5-131](#)

Name: **TPUCountChan(chan)**

Description: Change the TPU pin used for the TxBASIC COUNT command to the pin *chan*. COUNT defaults to channel 4. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example:
extension TPUCountChan
oldchan = TPUCountChan(6)

See Also: COUNT command on [page 5-26](#)

Name: **TPUGetTCR1()**

Description: This function returns the value of the TPU's Timer#1 frequency in Hz. The value should be the system freq. (from SimGetFSys) / 4.

Arguments: none

Returns: **long** -
TCR1 value in hertz

Example:
extension TPUGetTCR1
print "TCR1 frequency = ",TPUGetTCR1()

See Also: SimGetFSys on [page 5-126](#), TPUAccumPeriod on [page 5-129](#), TPUAccumPulseWidth on [page 5-130](#), TPUSetupPWM on [page 5-135](#), TPUChangePWM on [page 5-131](#)

Name: TPUoff()

Description: Turn Time Processor Unit off. The TPU is normally on when TxBASIC begins. This command saves about a milliamp of current. To turn the power back on, use TPUon().

Arguments: None

Returns: Nothing

Example:

```
extension TPUoff
TPUoff()           // Time Processor Unit now powered off
```

See Also: TPUon

Name: TPUon()

Description: Turn Time Processor Unit power on. The TPU is normally on when TxBASIC begins. This extension is used after TPUoff() has turned off the TPU.

Arguments: None

Returns: Nothing

Example:

```
extension TPUon
TPUon           // Time Processor Unit now powered up
```

See Also: TPUoff

Name: TPUPeriodChan(chan)

Description: Change the TPU pin used for the TxBASIC PERIOD command to the pin *chan*. PERIOD defaults to channel 4. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: the previous TPU channel used for this function.

Example:

```
extension TPUPeriodChan
print "The previous TPU line was ", TPUPeriodChan(15)
```

See Also: PERIOD command on [page 5-70](#)

Name: TPUSdiClkChan(chan)

Description: Change the TPU pin used for the TxBASIC SDI command clock to the pin *chan*. Default pin for SDI clock output signal is TPU channel 5. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example: extension TPUSdiClkChan
print "The previous TPU line was ", TPUSdiClkChan(15)

See Also: SDI command on [page 5-86](#), TPUSdiDatChan on [page 5-133](#), TPUSdiLchChan on [page 5-133](#)

Name: **TPUSdiDatChan(chan)**

Description: Change the TPU pin used for the TxBASIC SDI command data to the pin **chan**. Default pin for SDI data input is TPU channel 7. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example: extension TPUSdiDatChan
print "The previous TPU line was ", TPUSdiDatChan(15)

See Also: SDI command on [page 5-86](#), TPUSdiClkChan on [page 5-132](#), TPUSdiLchChan on [page 5-133](#)

Name: **TPUSdiLchChan(chan)**

Description: Change the TPU pin used for the TxBASIC SDI command latch to the pin **chan**. Default I/O pin for SDI latch out is TPU channel 3. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example: extension TPUSdiLchChan
print "The previous TPU line was ", TPUSdiLchChan(15)

See Also: SDI command on [page 5-86](#), TPUSdidatChan on [page 5-133](#), TPUSdiClkChan on [page 5-132](#)

Name: **TPUSdoClkChan(chan)**

Description: Change the TPU pin used for the TxBASIC SDO command clock to the pin **chan**. Default I/O pin for SDO clock out signal is TPU channel 5. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example: extension TPUSdoClkChan on [page 5-133](#)
print "The previous TPU line was ", TPUSdoClkChan(15)

See Also: SDO command, TPUSdoDatChan, TPUSdoLchChan

Name: **TPUSdoDatChan(chan)**

Description: Change the TPU pin used for the TxBASIC SDO command data to the pin **chan**. Default I/O pin for SDO clock out signal is TPU channel 8. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example: extension TPUDatClkChan
print "The previous TPU line was ", TPUSdoDatChan(15)

See Also: SDO command on [page 5-88](#), TPUSdoClkChan on [page 5-133](#), TPUSdoLchChan on [page 5-134](#)

Name: **TPUSdoLchChan(chan)**

Description: Change the TPU pin used for the TxBASIC SDO command latch to the pin **chan**. Default I/O pin for SDO clock out signal is TPU channel 2. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function.

Example: extension TPUSdoLchChan
print "The previous TPU line was ", TPUSdoLchChan(15)

See Also: SDO command on [page 5-88](#), TPUSdoClkChan on [page 5-133](#), TPUSdoDatChan on [page 5-134](#)

Name: **TPUToneChan(chan)**

Description: Change the TPU pin used for the TxBASIC Tone command to the pin **chan**. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: the previous TPU channel used for this function.

Example: extension TPUToneChan
print "The previous TPU line was ", TPUToneChan(15)

See Also: TONE command on [page 5-103](#)

Name: **TPUSetupPWM(chan, pwmhi, pwmpcr, priority)**

Description: This function set up a TPU channel as a pulse width modulated output channel. The channel will continue to output a signal till disabled by the priority being set to zero. The output will be low for pwmpcr - pwmhi cycles. This routine cannot be used on the TPU channel being used by USEND (defaults to TPU channel 13) or UGET (defaults to TPU channel 14).

Arguments: **int** -
1) TPU channel number
2) High time in TCR1 cycles [call TPUGetTCR1() for current value.]
3) Period time in TCR1 cycles.
4) The interrupt service priority. 0 is disabled, 1 is low, 2 is medium, and 3 is high.

Returns: none

Example:
extension TPUSetupPWM
TPUSetupPWM(8, Ptime, Pper, 2)

See Also: TPUChangePWM on [page 5-131](#), TPUGetTCR1 on [page 5-131](#)

Name: **TPUUgetChan(chan)**

Description: Change the TPU pin used for the TxBASIC UGET command to the pin *chan*. UGET defaults to channel 14. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function or -1 if it can't change the channel.

Example: extension TPUUgetChan
print "The previous TPU line was ", TPUUgetChan(15)

See Also: UGET command on [page 5-105](#)

Name: **TPUUseChan(chan)**

Description: Change the TPU pin used for the TxBASIC USEND command to the pin *chan*. USEND defaults to channel 13. The default TPU channel assignment for this extension is reestablished every time TxBASIC resets or when a program ends.

Arguments: **int** - TPU Channel number

Returns: **int** - the previous TPU channel used for this function or -1 if it can't change the channel.

Example: extension TPUUsendChan
 print "The previous TPU line was ", TPUUsendChan(15)

See Also: USEND command on [page 5-106](#)

Name: **TSerByteAvail(chan)**[TPU channels 0 to 15 only]

Description: This function returns the number of bytes available in the buffer of the specified channel. This function can also be used to return the number of characters remaining to be transmitted from an output buffer. This routine cannot be used on the TPU channel being used by USEND (defaults to TPU channel 13) or UGET (defaults to TPU channel 14). This routine will exit without acting unless the channel it accesses has been opened with TSerOpen. When TSerClose has been called for a channel, this routine will again exit without acting.

Arguments: **int** -
 TPU channel number.

Returns: **int** -
 returns 0 if the buffer is empty and non zero if the buffer has characters available.

Example:

```
EXTENSION TSerByteAvail
while (TSerByteAvail(7) = 0)
wend
```

See Also: TSerGetByte on [page 5-137](#), TSerInFlush on [page 5-137](#)

Name: **TSerClose(chan)**[TPU channels 0 to 15 only]

Description: This function closes the TPU serial port specified as chan. You will not be able to close channels 13 or 14. They must be left open for UGET and USEND. This routine cannot be used on the TPU channel being used by USEND (defaults to TPU channel 13) or UGET (defaults to TPU channel 14)

Arguments: **int** -
 TPU channel number.

Returns: **int** -
 the function returns 0 if successful and 1 if not successful.

Example:

```
extension TSerClose
TSerClose(7)
TSerClose(8)
```

See Also: TSerOpen on [page 5-137](#)

Name: **TSerGetByte(chan)** [TPU channels 0 to 15 only]

Description: This function gets the next byte from the channels input queue or sits and waits for the next byte if no bytes are available. Usually call this function only after checking `TSerByteAvail()`. This routine cannot be used on the TPU channel being used by `USEND` (defaults to TPU channel 13) or `UGET` (defaults to TPU channel 14). This routine will exit without acting unless the channel it accesses has been opened with `TSerOpen`. When `TSerClose` has been called for a channel, this routine will again exit without acting.

Arguments: **int** -
TPU channel number.

Returns: **int** -
returns the next available byte from the input queue.

Example:

```
extension TSerGetByte
A = TSerGetByte()
```

See Also: `TSerInFlush` on [page 5-137](#), `TSerPutByte` on [page 5-138](#)

Name: **TSerInFlush(chan)** [TPU channels 0 to 15 only]

Description: This function flushes the channel's input queue (or output queue depending on whether the channel was opened as in input or output channel) and discards any characters in this queue. This works for both the Input and Output channels. This routine cannot be used on the TPU channel being used by `USEND` (defaults to TPU channel 13) or `UGET` (defaults to TPU channel 14). This routine will exit without acting unless the channel it accesses has been opened with `TSerOpen`. When `TSerClose` has been called for a channel, this routine will again exit without acting.

Arguments: **int** -
TPU channel number.

Returns: none

Example:

```
extension TSerInFlush
TSerInFlush(7)
TSerInFlush(8)
```

See Also: `TSerByteAvail` on [page 5-136](#), `TSerGetByte` on [page 5-137](#), `TSerPutByte` on [page 5-138](#)

Name: **TSerOpen(chan, priority, outflag, buffer, qsize, baud, parity, bits, stop)** [TPU channels 0 to 15 only]

Description: This functions opens a TPU channel for serial I/O. You cannot open channels 13 or 14 because they are opened when `TXBasic` starts. This routine cannot be used on the TPU channel being used by `USEND` (defaults to TPU channel 13) or `UGET` (defaults to TPU channel 14).

- Arguments: The following 6 arguments MUST be provided:
- 1) **int** - TPU channel number. The second serial port on the TT8 uses channel 13 for transmit and channel 14 for receive.
 - 2) **int** - this is the channel interrupt service priority flag with 0 = disable, 1 = low, 2 = middle, and 3 = high priority.
 - 3) **int** - the outflag determines whether the channel is input or output with 0 = input and 1 = output.
 - 4) **int** - this argument is only a place holder for the C language function called by this extension.
 - 5) **long** - the qsize must be an integral power of 2 (2,4,8,16,32,...). A good general value for this would be 256.
 - 6) **long** - the requested baud rate

The remaining 3 argument variables are optional. If not specified, they default to parity = none, bits = 8, and stopbits = 1.

- 7) **int** - parity is an alphabetical character as follows:
 - 'N' = None
 - 'O' = Odd
 - 'E' = Even
- 8) **int** - this argument is the number of bits in the data stream. This argument can be from 1 bit to 14 bits in length.
- 9) **int** - this argument is the number of stop bits. This argument can be 1 or 2 bits long.

Returns: **int** - the function returns 0 if successful and 1 if not successful.

Example:

```
extension TSerOpen
if TSerOpen(8,2,1,0,128,57600) = 1 err = 1 //:goto error
if TSerOpen(7,2,0,0,128,57600) = 1 err = 2 //:goto error
```

See Also: TSerResetBaud on [page 5-139](#), TSerClose on [page 5-136](#)

Name: **TSerPutByte(chan, data)**[TPU channels 0 to 15 only]

Description: This function writes a byte of data the the channels output queue. This routine cannot be used on the TPU channel being used by USEND (defaults to TPU channel 13) or UGET (defaults to TPU channel 14). This routine will exit without acting unless the channel it accesses has been opened with TSerOpen. When TSerClose has been called for a channel, this routine will again exit without acting.

Arguments: **int** -
1) TPU channel number
2) byte to be transmitted

Returns: none

Example:
extension TSerPutByte
for n=&h30 to &h7e
 TSerPutByte(8,n)
next n

See Also: TSerInFlush on [page 5-137](#), TSerGetByte on [page 5-137](#)

Name: **TSerResetBaud(chan, baud)**[TPU channels 0 to 15 only]

Description: This function attempts to change the current baud rate for the selected channel to the new baud rate specified by the argument baud. This routine cannot be used on the TPU channel being used by USEND (defaults to TPU channel 13) or UGET (defaults to TPU channel 14). This routine will exit without acting unless the channel it accesses has been opened with TSerOpen. When TSerClose has been called for a channel, this routine will again exit without acting.

Arguments: **int** -
TPU channel number.

long -
the expected baud rate.

Returns: **long** -
returns the actual baud rate set by the function.

Example:
extension TSerResetBaud
bd = TSerResetBaud(8,19200)

See Also: TSerOpen on [page 5-137](#)

Section 6 - Hardware and Interface Specifications

Getting Started

This section shows you the detailed specifications needed for connecting devices to the Tattletale. First time users will generally work with a Model 8 that has an I/O-8 prototyping board on top (included in the development kit). If you bought the deluxe kit you will also have a PR-8 prototyping board in the development kit (it's the 5 x 7 inch board).

Once you are comfortable using TxBASIC and TxTools, you are ready to look at the Model 8 hardware in more detail and start designing your own hardware applications.

Tattletale Model 8 Connectors

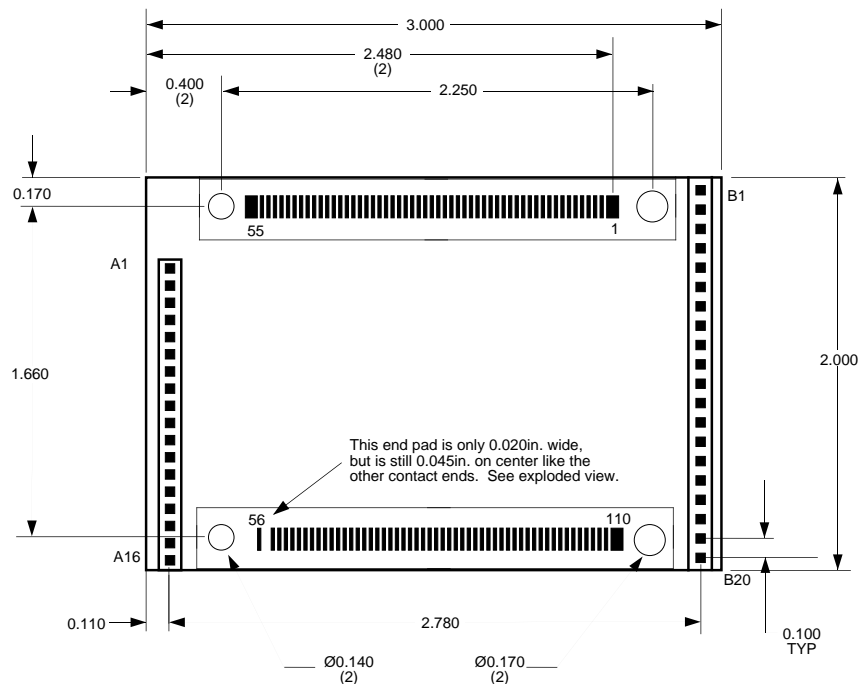
Pin and Socket Connector Specifications

These connectors mate with the connectors on the Model 8. Samtec part numbers are shown.

Mating connectors for the Model 8

16 pin "A" connector	TSW-116-07-SS
20 pin "B" connector	TSW-120-07-SS

Connections A1-A16 are sockets for 0.025in. square pins expressed on the top only. Connections B1-B20 are sockets for 0.025in. square pins expressed on the top only. Square connectors are 0.290 inches high. Pins are on 0.1in. centers. Allow 0.180in. clearance on either side of board for components.



NOTE: All dimensions are in inches.

Figure 6-1: Model 8 Dimensions

SquishyBus Connector Specifications

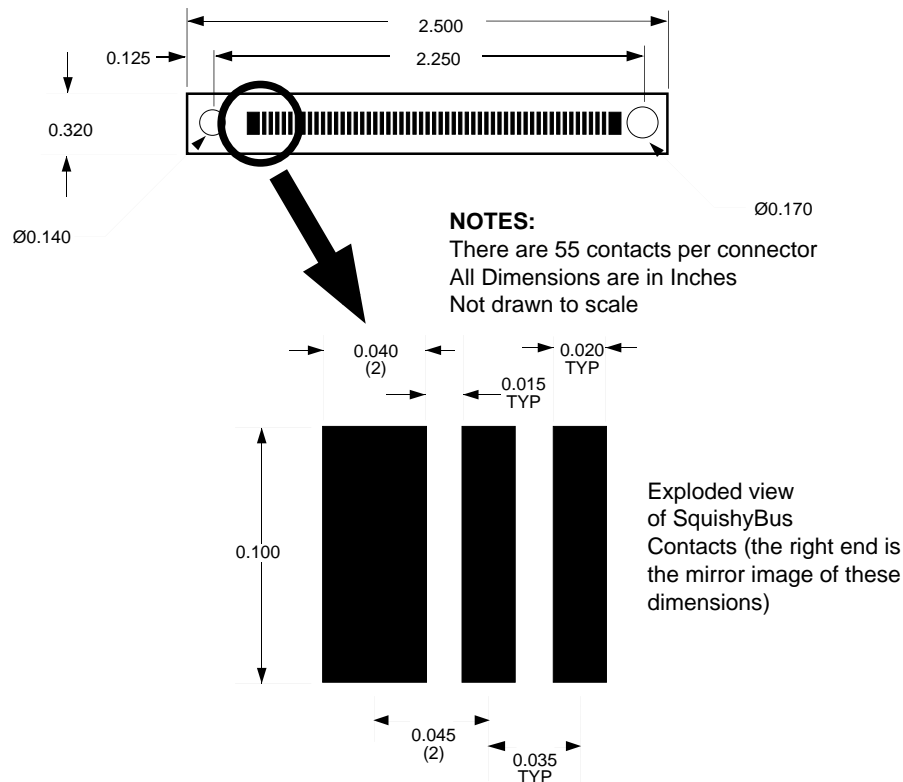


Figure 6-2: SquishyBus Dimensions

Connections 1-110 are surface contacts expressed on both sides of the Tattletale board. There are 55 contacts per SquishyBus connector. The spacing between contacts is 0.015in. Contact widths are 0.020in. except ends which are 0.040in. The connectors are 0.380 high.

The dimensions of the Model 8 printed circuit board are shown in Figure 6-1. The footprint is 2in. x 3in. Including the socket connectors and ICs, the thickness of the populated Model 8 board is 0.55in. The socket connectors are designed to mate with 0.025in. square pins. While socket connectors are located only on the top side of the board, the gold pads for use with elastomeric expansion connectors (SquishyBus) are located on both sides of the PR-8 board.

Mounting the Model 8 to the Prototyping Boards

Four mounting holes are available for securing the Model 8 in place. Two of the mounting holes have a 0.140in. diameter; and two have a 0.170in. diameter. The differing diameters help to assure proper alignment of the elastomeric connectors. The Model 8 must be secured in place through all four holes with machine screws and nuts before putting it into permanent service. The connectors alone do not provide secure connection. Expansion with elastomeric interconnects (SquishyBus) requires no spacers as the elastomeric interconnects themselves serve as spacers.

NOTE: If you are using the pin & socket connectors in your finished system (like with the I/O-8 board), you should use spacers between each hole and the relative securing point. Failure to do so may result in twisted or bent pin and socket connectors.

The gold pads are arranged in rows where the outside pads have a larger width than the inside pads. The outside pads have a width of 0.040in. and the inside pads have a width of 0.020in. Spacing between the outside pads and inside pads is 0.045in. center to center. Spacing between inside pads is 0.035in. center to center. Elastomeric interconnects should be selected accordingly. Interconnects are available in various heights to accommodate your external hardware, an example of an elastomeric interconnect is the Fujipoly's part number 0940020 which is 0.250in high. Elastomeric interconnects are available from Onset.

NOTICE

The Model 8 is a multiple layered board. Do not attempt to cut or lift traces on the board, except designated ones, as lower layers may be damaged.



If you purchased the deluxe development kit or the PR-8 board, you have received a pair of elastomeric interconnects, 4 nylon screws and nuts and a hex nut driver.

Prototyping Board Details

The I/O-8 and PR-8 prototyping boards have convenient connection points for the communication cable (UART connection), and power.


Battery Power

During your initial development you will need a DC power supply with an output in the range of 7 to 15V that can provide a current of about 300mA. *We strongly recommend a current limited supply.* The I/O-8 and PR-8 prototyping boards have DC power jacks on the boards.


NOTICE


The Tattletale can be severely damaged by reversing the power supply or battery connections. **DO NOT** reverse the connections or you will void the warranty.

Before connecting a power supply to the DC power jack, make sure that the center of the tip is negative and the sleeve is positive.



Sleeve + → ← - Tip Center

Correct DC Power Jack Polarity

When no connector is in the DC power jack, the AUX line is connected to the BAT line. When a connector is inserted in the jack, it breaks this connection and supplies power directly to the BAT line. The AUX line is left floating. So, you could have a battery connected to the AUX line. Then when you insert the plug (from some other power source), it will disconnect your battery and supply the Tattletale from the other source. These drawings show how the plug works schematically. Figure 6-3 shows the jack with nothing plugged in. The AUX and BAT lines are connected at the arrow:



Figure 6-3: DC Power Jack w/o Connector Plugged in

Now, with a plug inserted, the bottom line (which is a spring) is forced away from the AUX line as shown in Figure 6-4:

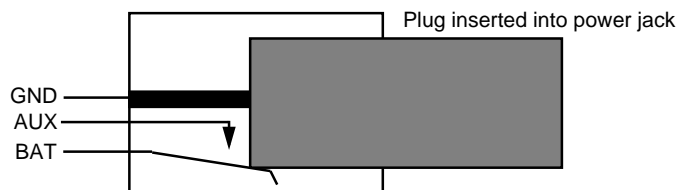


Figure 6-4: DC Power Jack with Connector Plugged in

The UART

The Model 8 has a default baud rate of 9600 baud, eight data bits, no parity and one stop bit. Interface connections are normally made using the PC-3.5 communication cable. The only Tattletale connections to worry about are TXD, RXD and GND; however, your computer may require other connections. Because many computers require it, the PC-3.5 cable ties several pins together, as shown in Figure 6-5.

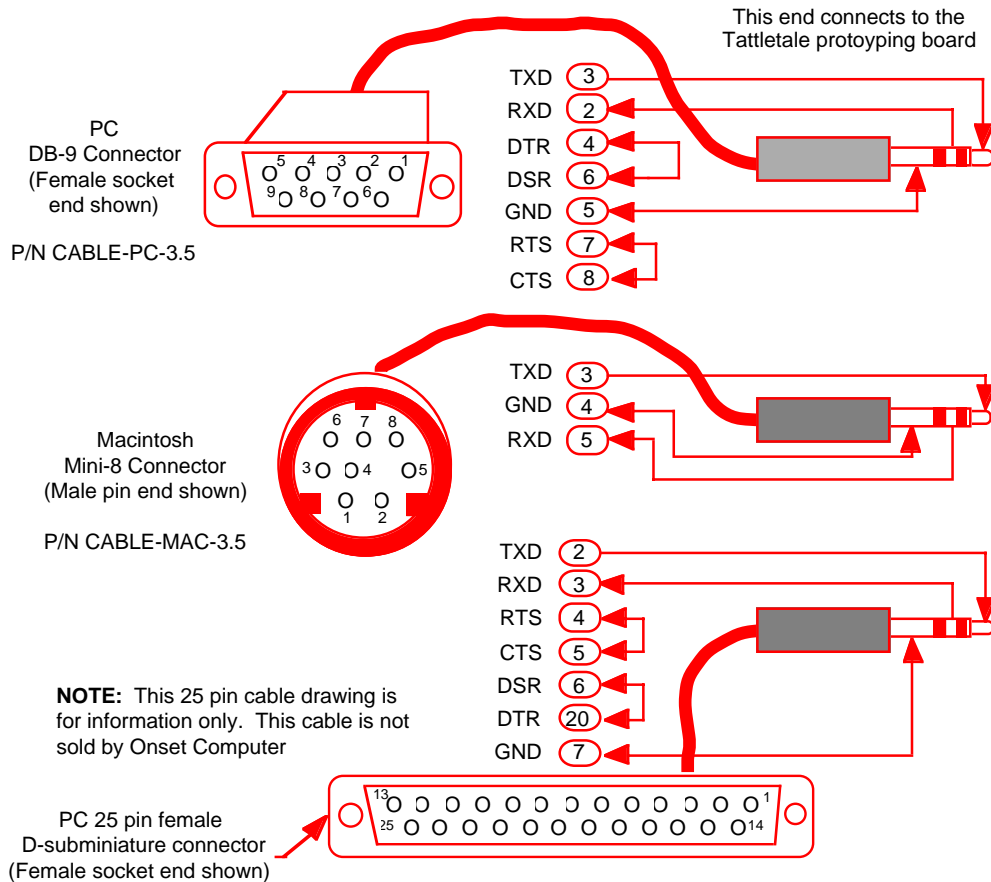


Figure 6-5: Communication Cables

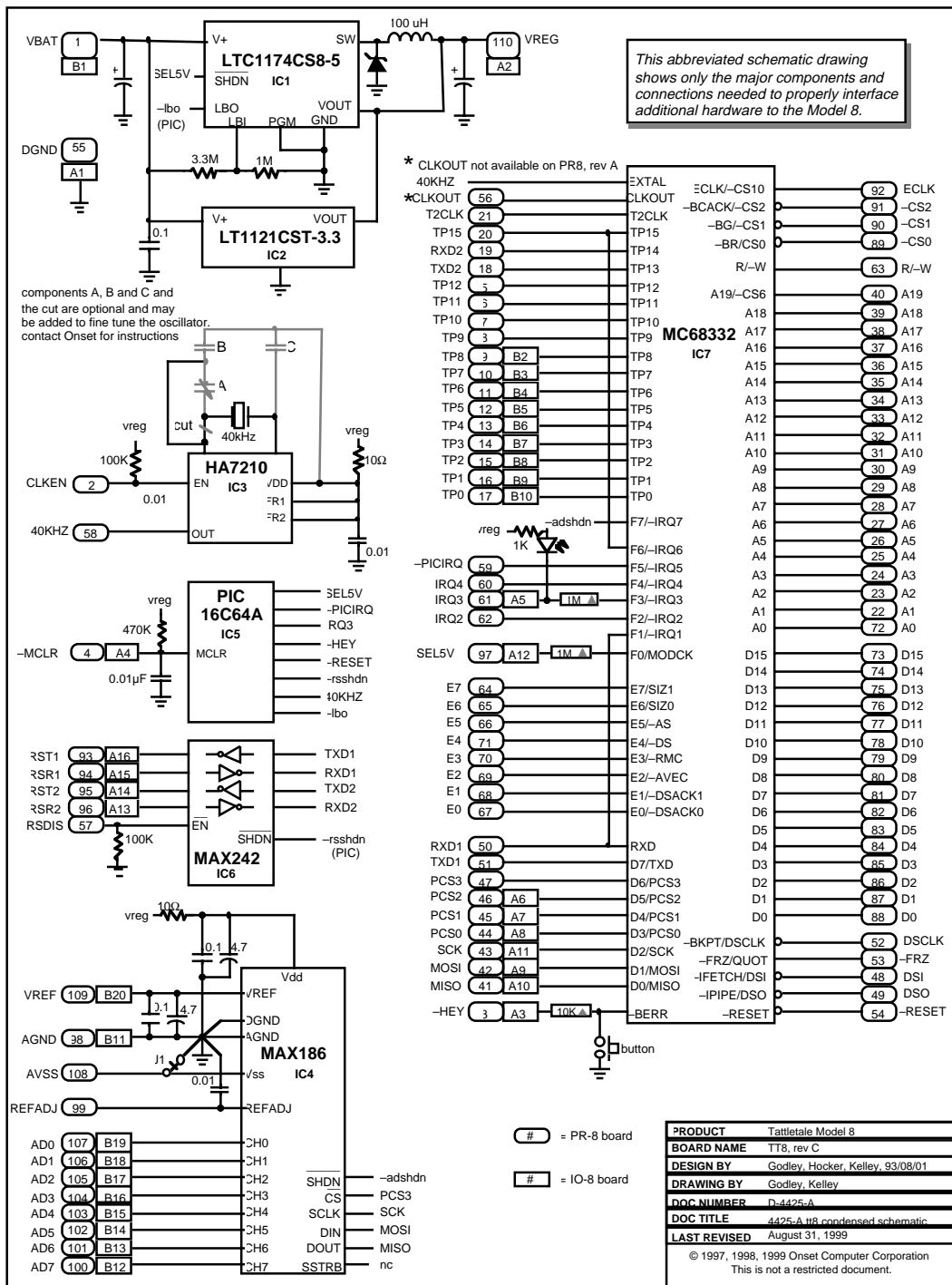


Figure 6-6: Schematic of the Model 8

Tattletale Model 8 Connections and Specifications

A major design goal for the Model 8 was minimum size. To achieve this, parts are mounted on both sides of the board. Table 6-1 shows the Model 8 specifications. Refer to Table 6-2 and Table 6-3 for specific pin functions.

Table 6-1: Model 8 Specifications

Size (inches)	2 x 3 x 0.5
Weight (oz.)	1
Processor	68332
Data capacity (RAM)	256K/1M
Additional capacity	PCMCIA
Flash EEPROM	256K
A-D converter	12-bit
Analog channels	8
Max sampling rate (Hz)	100K
Digital I/O lines	up to 25
Count channels	up to 25
Minimum current	<200 μ A typical
Peak current	150mA
Main UART baud (default) at RS-232 Levels:	9600
TPU UART baud rates (others available):	The 14 TPU lines can be set to any standard rate up to 500K
Serial EEPROM (bytes)	7190
Voltage input	7 to 15V
Battery RAM backup	No
Real-time clock	Hardware
Programming languages	C, TxBASiC
Operating temperature range	-40 to +85° C
Relative humidity range	0 - 95% non-condensing

Operating Temperature Range

Model 8 components are specified to operate over a temperature range of -40°C to +85°C with the following exceptions. The switch used to enter the Background Debugging Mode (BDM) during the power up sequence has an operating range of -20°C to +70°C. The Light Emmiting Diode (LED) used to indicate a low signal on IRQ3 (pin 61 on PR-8, pin A-5 on IO-8) has an operating range of -30°C to +85°C.

Table 6-2: I/O-8 Pin Functions

I/O-8 Pin#	Signal	Function Description
A1	DGND	The digital ground. This is the negative return for the digital circuitry
A2	VREG	Positive supply
A3	-HEY	I/O with Interrupt on change pin (PIC)
A4	-MCLR	Master clear (active low reset); use to reset the Model 8
A5	-IRQ3	Interrupt request (level 3)
A6	PCS2	Peripheral chip select
A7	PCS1	Peripheral chip select
A8	PCS0	Peripheral chip select
A9	MOSI	Master-Out Slave-In
A10	MISO	Master-In Slave-Out
A11	SCK	QSPI serial clock
A12	SEL5V	Clock mode select
A13	RSR2	RS-232 receive (Jack 2)
A14	RST2	RS-232 transmit (Jack 2)
A15	RSR1	RS-232 receive (Jack 1)
A16	RST1	RS-232 transmit (Jack 1)
B1	VBAT	The main battery supply
B2	TP8	TPU channel 8
B3	TP7	TPU channel 7
B4	TP6	TPU channel 6
B5	TP5	TPU channel 5
B6	TP4	TPU channel 4
B7	TP3	TPU channel 3
B8	TP2	TPU channel 2
B9	TP1	TPU channel 1

Table 6-2: I/O-8 Pin Functions (Continued)

I/O-8 Pin#	Signal	Function Description
B10	TP0	TPU channel 0
B11	AGND	The analog ground
B12	AD7	A/D channel 7
B13	AD6	A/D channel 6
B14	AD5	A/D channel 5
B15	AD4	A/D channel 4
B16	AD3	A/D channel 3
B17	AD2	A/D channel 2
B18	AD1	A/D channel 1
B19	AD0	A/D channel 0
B20	VREF	The reference voltage for the A/D, or output of the reference buffer amplifier

Table 6-3: PR-8 Pin Functions

PR-8 Pin#	Signal	Function Description
1	VBAT	The main battery supply
2	CLKEN	Clock enable
3	-HEY	Interrupt on change pin (PIC); bus error signal to 332
4	-MCLR	Master clear (reset) input/prog voltage input to PIC; active low reset
5	TP12	TPU channel 12
6	TP11	TPU channel 11
7	TP10	TPU channel 10
8	TP9	TPU channel 9
9	TP8	TPU channel 8
10	TP7	TPU channel 7
11	TP6	TPU channel 6

Table 6-3: PR-8 Pin Functions (Continued)

PR-8 Pin#	Signal	Function Description
12	TP5	TPU channel 5
13	TP4	TPU channel 4
14	TP3	TPU channel 3
15	TP2	TPU channel 2
16	TP1	TPU channel 1
17	TP0	TPU channel 0
18	TXD2	TPU UART transmit
19	RXD2	TPU UART receive
20	TP15	TPU channel 15
21	T2CLK	TPU clock in
22	A1	CPU Address Line (A1)
23	A2	CPU Address Line (A2)
24	A3	CPU Address Line (A3)
25	A4	CPU Address Line (A4)
26	A5	CPU Address Line (A5)
27	A6	CPU Address Line (A6)
28	A7	CPU Address Line (A7)
29	A8	CPU Address Line (A8)
30	A9	CPU Address Line (A9)
31	A10	CPU Address Line (A10)
32	A11	CPU Address Line (A11)
33	A12	CPU Address Line (A12)
34	A13	CPU Address Line (A13)
35	A14	CPU Address Line (A14)
36	A15	CPU Address Line (A15)
37	A16	CPU Address Line (A16)
38	A17	CPU Address Line (A17)

Table 6-3: PR-8 Pin Functions (Continued)

PR-8 Pin#	Signal	Function Description
39	A18	CPU Address Line (A18)
40	A19	CPU Address Line (A19)
41	MISO	Master-In Slave-Out (QSPI)
42	MOSI	Master-Out Slave-In (QSPI)
43	SCK	QSPI serial clock
44	PCS0	QSPI chip select
45	PCS1	QSPI chip select
46	PCS2	QSPI chip select
47	PCS3	QSPI chip select
48	DSI	Development serial in
49	DSO	Development serial out
50	RXD1	SCI receive data
51	TXD1	SCI transmit data
52	DSCLK	Development serial clock
53	–FRZ	Freeze
54	–RESET	Reset
55	DGND	Digital ground
56	NC	No connection
57	RSDIS	RS-232 driver disable
58	40KHZ	40KHz clock out
59	–PICIRQ	Interrupt request
60	–IRQ4	Interrupt request (level 4)
61	–IRQ3	Interrupt request (level 3)
62	–IRQ2	Interrupt request (level 2)
63	R/–W	Read/Write
64	E7	E Port 7
65	E6	E Port 6

Table 6-3: PR-8 Pin Functions (Continued)

PR-8 Pin#	Signal	Function Description
66	E5	E Port 5
67	E0	E Port 0
68	E1	E Port 1
69	E2	E Port 2
70	E3	E Port 3
71	E4	E Port 4
72	A0	CPU Address Line (A0)
73	D15	CPU Data Line (D15)
74	D14	CPU Data Line (D14)
75	D13	CPU Data Line (D13)
76	D12	CPU Data Line (D12)
77	D11	CPU Data Line (D11)
78	D10	CPU Data Line (D10)
79	D9	CPU Data Line (D9)
80	D8	CPU Data Line (D8)
81	D7	CPU Data Line (D7)
82	D6	CPU Data Line (D6)
83	D5	CPU Data Line (D5)
84	D4	CPU Data Line (D4)
85	D3	CPU Data Line (D3)
86	D2	CPU Data Line (D2)
87	D1	CPU Data Line (D1)
88	D0	CPU Data Line (D0)
89	-CS0	Chip select
90	-CS1	Chip select
91	-CS2	Chip select
92	ECLK	Outputs 1/8 system clock (can also be used as chip select)

Table 6-3: PR-8 Pin Functions (Continued)

PR-8 Pin#	Signal	Function Description
93	RST1	RS-232 transmit (jack 1)
94	RSR1	RS-232 receive (jack 1)
95	RST2	RS-232 transmit (jack 2)
96	RSR2	RS-232 receive (jack 2)
97	SEL5V	Clock mode select
98	AGND	The analog ground
99	REFADJ	A/D Reference Adjust
100	AD7	A/D channel 7
101	AD6	A/D channel 6
102	AD5	A/D channel 5
103	AD4	A/D channel 4
104	AD3	A/D channel 3
105	AD2	A/D channel 2
106	AD1	A/D channel 1
107	AD0	A/D channel 0
108	AVSS	Can be negative reference for A/D in bipolar mode or GND
109	VREF	The reference voltage for the A/D, or 4.096 V out
110	VREG	Positive supply

Digital I/O Line Connections

The Model 8 has up to 25 digital I/O lines (see Table 6-2 and Table 6-3 for the I/O line and pin numbers). Each of these lines can be used as an input or an output.

For additional functionality, we have designed many of the I/O lines with the ability to be used for special functions (all of which are completely optional). Table 6-4 shows a list of all the I/O lines and which ones have a special function and exactly what that function is. Table 6-5 shows a detailed description of each of the optional special functions.

Example of using the Special Digital I/O Line Functions

For this example we are putting the Tattletale to sleep until an external trigger wakes the Tattletale and causes the program to continue. If we connect I/O line 1 to the external trigger and enter the following lines of code into our program, the Tattletale will stay asleep until the external trigger on line 1 goes high, at which time the program will continue with the other commands on the same line as the sleep command. Once the external trigger goes high the program will continue with the code after the label “Wakeup”. The program would read:

```

REM Put Tattletale to sleep until external trigger goes high
label1: sleep 1000 goto wakeup: REM Will sleep until I/O line 1 goes high
      goto label1
wakeup: Continue with your program code

```

NOTE: Please notice that there is not a colon between the sleep 1000 command and the goto command. Refer to the SLEEP command on [page 5-91](#) for detailed information on using the sleep command.

Table 6-4: Digital I/O Lines with Special Functions

I/O-8 Pin#	PR-8 Pin#	I/O Line	Special Function
B9	16	I/O line 1 TPU CH 1	SLEEP wakeup
B8	15	I/O line 2 TPU CH 2	SDO latch
B7	14	I/O line 3 TPU CH 3	SDI latch
B6	13	I/O line 4 TPU CH 4	PERIOD & COUNT input
B5	12	I/O line 5 TPU CH 5	SDI and SDO clock
B4	11	I/O line 6 TPU CH 6	TONE & DTOA Output
B3	10	I/O line 7 TPU CH 7	SDI data line
B2	9	I/O line 8 TPU CH 8	SDO data line
A14	95	I/O line 13 TPU CH 13	USEND output
A13	96	I/O line 14 TPU CH 14	UGET input

Important Information before using the Digital I/O Lines

The I/O lines (TPU lines) have identical characteristics and can be set up in any combination of inputs and outputs, and the sense and direction of a line can be reprogrammed as often as desired. Care should be taken to ensure that all lines driven by an external source are defined as inputs. Not doing so could lead to high current drain and possible damage to the logger. All I/O lines are set to input at power-up to eliminate this potential conflict. Refer to Table 6-6 for digital I/O specifications.

The current drain caused by a CMOS input is a strong function of voltage applied to the input, dropping to near zero when the input voltage is within a diode drop of GND or VREG.

NOTE: Floating inputs can cause high current drain; thus unused lines should be pulled to GND or VREG through 1 Meg (max.) resistors or set to outputs. The digital I/O lines should never be exposed to voltages above the board's positive supply (VREG) or below the GND level, as this could cause latchup on the board. We recommend that all digital interface circuitry be powered from VREG, and that all external circuitry powered from a separate supply be connected with open collector circuits to eliminate any chance of applying out-of-range voltages to the board.

Table 6-5: Detailed Descriptions of the Special I/O Line Functions

Special Function and Description
<p>External Sleep Wakeup: If a SLEEP command is followed by commands (not separated from the SLEEP command by a colon), these commands will be executed if I/O line 1 goes high before the normal time-out of the SLEEP command. Refer to the SLEEP command on page 5-91 for details.</p>
<p>SDO Latch Line: This line has an alternate use as the latch line in the TxBASIC SDO command. For details of the hardware connections for SDO, refer to the SDO command on page 5-88 for details.</p>
<p>SDI Latch Line: This line has an alternate use as the latch line in the TxBASIC SDI command. For details of the hardware connections for SDI, refer to the SDI command on page 5-86 for details.</p>
<p>SDI and SDO Clock Line: This line has an alternate use as the clock line in the TxBASIC SDI and SDO commands. For details of the hardware connections for SDO and SDI, refer to the SDI and SDO commands on page 5-86 and page 5-88 for details.</p>

Table 6-5: Detailed Descriptions of the Special I/O Line Functions (Continued)

Special Function and Description
<p>Aux UART Input: This line is treated by the TxBASIC UGET command as an asynchronous serial input line. The input to this line should be CMOS levels, not exceeding VREG or dropping below ground. The sense of this signal is inverted from the normal sense of RS-232 lines, with the marking state expected to be HIGH (the marking state, also known as the idle state, is +5V). Under no circumstances should this pin be exposed to a voltage below ground potential or above the positive supply. Many peripherals can connect to this input directly (CMOS levels, inverted logic).</p>
<p>Aux UART Output: This line can be used as an asynchronous serial output line by using the TxBASIC USEND command. The signal output levels are 0 to VREG, inverted from the normal sense of RS-232 lines, with the transmitted marking state HIGH. Under no circumstances should this pin be exposed to a voltage below ground potential or above the positive supply (VREG). Many peripherals can connect to this input directly (CMOS levels, inverted logic).</p>
<p>SDI Data Line (Shift Register Input): This line has an alternate use as the data input line in the TxBASIC SDI command. For details of the hardware connections for SDI, refer to the SDI command on page 5-86 for details.</p>
<p>SDO Data Line (Shift Register Output): This line can be either an input or output. It is also used along with I/O line 5 and I/O line 2 in the SDO command. For details of the hardware connections for SDO, refer to the SDO command on page 5-88 for details.</p>
<p>TONE Output Line or DTOA Output Line: This line can be used as an output line in either the TONE or DTOA command. Refer to the TONE command on page 5-103 and the DTOA command on page 5-36 for details.</p>
<p>PERIOD, COUNT Input: The TxBASIC commands PERIOD and COUNT use I/O line 4 for their input. Refer to the PERIOD command on page 5-70 and the COUNT command on page 5-26 for details.</p>

Table 6-6: Digital I/O Line Specifications (from Motorola)

Item	Test Condition	Min.	Max.	Units
Input "High" Voltage	—	2.0	5.0	Volts
Input "Low" Voltage	—	0	0.8	Volts
Output "High" Voltage	Sourcing 200 μ A	2.4	—	Volts
Output "High" Voltage	Sourcing 10 μ A	4.3	—	Volts
Output "Low" Voltage	Sinking 1.6mA	—	0.4	Volts
Input Capacitance	25°C, f=1 MHz	—	12.5	pF

Analog Input Connections and Specifications

12-Bit, 8-Channel A-D Converter

Connecting to the Analog Inputs:

The Model 8 includes a MAX186 A/D converter for handling analog inputs. The MAX186 is a 12 bit, 8 channel device which can operate in unipolar or bipolar mode. The MAX186 can also operate with either external or internal reference. The factory standard configuration in which the Model 8 is supplied is internal reference. The internal reference is 4.096V unipolar full scale and $\pm 2.048V$ in bipolar mode.

NOTE: Because a VSS and REFADJ are not brought out to the connectors, bipolar mode is not available while using the I/O-8 prototyping board; however, it is available while using the PR-8 board.

To use an external reference, simply provide your reference voltage to REFADJ or VREF. Using external reference, full scale can be as high as $V_{dd} + 50mV$. Sensors which are particularly appropriate to such ratiometric operation include strain gauges, thermistor bridges and other resistive divider circuits where the output voltage is proportional to the input voltage. None of the Model 8's channels are connected to a sensor (unless you soldered the thermistor and resistor to the board in **Section 2 -How to Connect and Setup the Model 8** and did not remove the components after testing the Tattletale).

Analog Inputs

The Model 8 has eight analog inputs to a 12-bit A-D converter. The data sheets on all the Model 8 IC's are in the appendix of this manual. Table 6-7 shows the pins on the converter strip used by the I/O-8 and the PR-8 boards.

Table 6-7: Analog Input Pin Specifications

I/O-8 Pin #	PR-8 Pin #	Analog Input Pin Function
---	39	Negative reference for the A-D in bipolar mode (AVSS)
B20	109	VREF (should be between 2.5V and 5V)
B11	98	Analog Ground (AGND)
B19	107	A-D chan 0
B18	106	A-D chan 1
B17	105	A-D chan 2
B16	104	A-D chan 3
B15	103	A-D chan 4
B14	102	A-D chan 5
B13	101	A-D chan 6
B12	100	A-D chan 7

Main UART

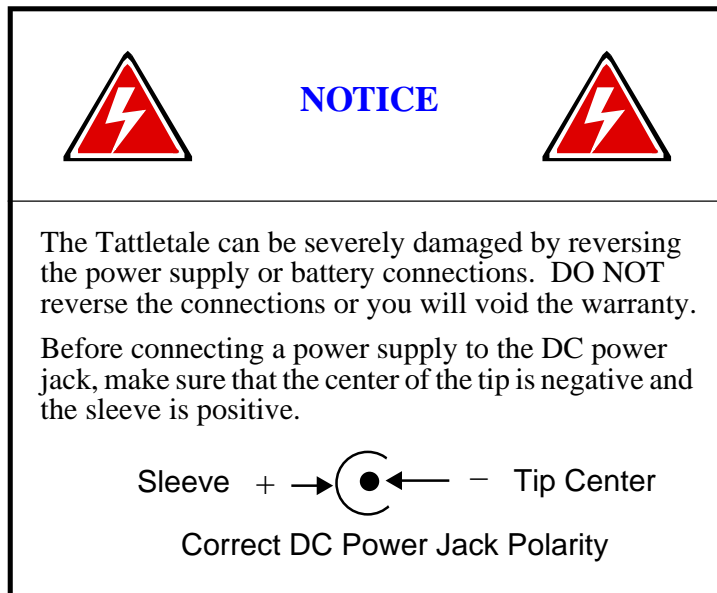
At power-up the hardware UART (at RS-232 levels) is programmed for 9600 baud, eight data bits, one stop bit and no parity. The 14 TPU lines can be set to any standard baud rate up to 500K.

The maximum software UART baud rates available at TTL levels are 300-500K. TPU I/O pins may be configured through software as either a UART Rx or Tx. The TPU handles all timing and buffering of serial transmissions and therefore achieves a CPU independence equivalent to a hardware UART.

Changing the Baud Rate

You can change the UART's baud rate by using the TxBASIC extension SerSetBaud. Please refer to [Section 5 - TxBASIC Command Reference](#) for additional information on using the command.

Power Supply Considerations



Battery connection points are available on the Tattletale (BAT+ is at pin B1, GND is at pin B11). The I/O-8 and PR-8 prototyping boards bring these lines to a standard DC power jack. The supply voltage should be between 7 and 15 volts.

The Model 8 operates at two levels of VREG. When awake and active, VREG is maintained at 5 Volts. In low power drain mode, Vreg is maintained at 3.3 Volts. In order to take full advantage of these two modes, the Model 8 utilizes two voltage regulators. The LTC1174CS8-5 switching regulator is the relevant regulator in 5V mode. The 1174 has an absolute maximum input supply voltage rating of 15V which must not be exceeded! In low power mode, the 1174 is shutdown. In this shutdown mode, the 1174 draws only 1 μ A. The LT1121CST-3.3 linear supply, which is always active, provides VREG in the low power mode. The 1174 will provide up to 340mA in the 5V mode. The 1121 can provide up to 200mA in the 3.3V mode.

Current Drain

In 5V mode, the bulk of the current drain will come from the MC68332. The 68332 will draw about 1mA per MHz, but may draw upwards of 70mA when performing tasks which are I/O intensive. By way of comparison, in the 5V mode the PIC and other components should draw less than 10mA. In 3.3V mode, the 68332 will draw about 20-30 μ A. Plan accordingly if you intend to use VREG for peripheral devices. The most important point to remember is that the current drain is largely dependent on functions you have the Model 8 perform, i.e., it is software dependent. For this reason, the only practical way to determine whether you can power your peripherals from VREG is through experimentation with your application and direct measurement.

The Tattletales typical current drain is a strong function of the program it is running (see Table 6-8). The current drain is low while waiting for a character from the UART. Be sure to disconnect the RS-232 cable from the Tattletale when it is not in use as the interface drivers take an additional 1 to 2mA.

Table 6-8: Current Drain Due to Various Commands

Program	Drain (typical)
Asleep // RS232, A-D & TPU off, LP Mode(1)	1.5mA
Label: GOTO Label // LP Mode(2)	25mA
Label: A=CHAN(0) :GOTO Label // LP Mode(2)	25mA
HYB	150 μ A
Refer to Section 4 - Using TxBASiC and Section 5 - TxBASiC Command Reference for TxBASiC information.	

Table 6-9: Current Drain Due while using Low Power Modes

TxBASiC Current Modes on the Tattletale Model 8			
LPMODE	A-D on	A-D off	System Clock
1	17.7mA	15.9mA	640 kHz, 1.28 MHz or 1.92 MHz
2	26.2mA	24.3mA	3.68 MHz
3	32.8mA	31.0mA	6.08 MHz
4	36.3mA	34.4mA	7.36 MHz
5	41.4mA	39.5mA	9.12 MHz
6	46.8mA	45.0mA	11.2 MHz
7	50.9mA	49.0mA	12.8 MHz
8	56.0mA	54.2mA	14.72 MHz
9	60.0mA	58.1mA	16.0 MHz
Command	A-D on	A-D off	System Clock
SLEEP	3.2mA	1.5mA	3.68 MHz
HYB	750 μ A	140 μ A	Off

A-D Converter Circuit

Figure 6-7 is a schematic of the A-D converter on the Model 8.

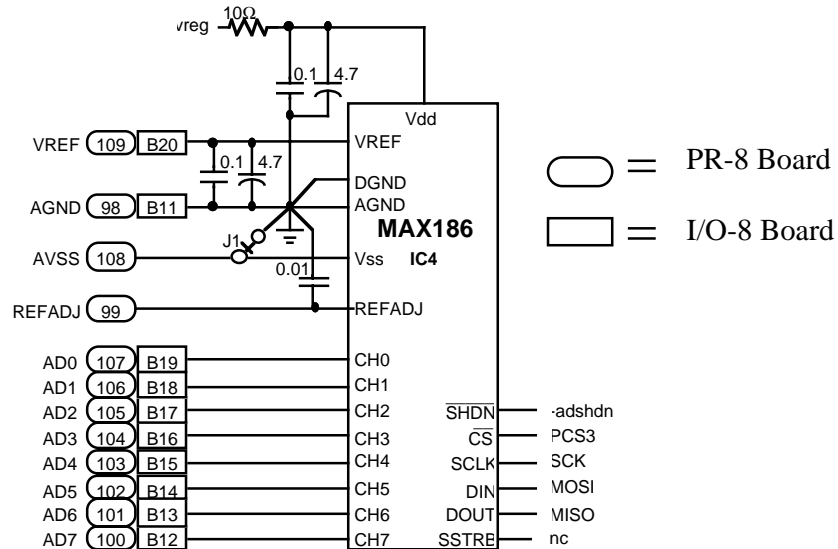


Figure 6-7: Diagram of A-D Regulator Circuit

RESET

The Model 8's pin A4 on the I/O-8, (pin 4 on the PR-8), is $\overline{\text{MCLR}}$, which acts as the TT8 reset.

Memory

RAM

Depending on the configuration of your Model 8, you may have 256K of static RAM, or 1M of static RAM. Naturally, static RAM can be used for variable and array storage, stack and heap maintenance, and program development.

Flash Memory

Depending on the configuration, your Model 8 may have 256K or 1M* of Flash EEPROM. The Flash parts can be programmed and erased quickly. Further, they have a near zero current drain while inactive.

* Not available as of this writing - call for further information.

QSM - Queued Serial Module

SCI - Serial Communications Interface

The SCI can be used to communicate with external devices and can be reached on both the IO-8 and PR-8 through UART jack 1 (RS-232(1)). The SCI UART operates at rates from 64 baud to 500 KBaud. It features advanced error detection, full duplex operation, selectable word length, receiver wakeup and parity generation and detection.

SIM - System Integration Module

System Clock

The 68332 derives its system clock from a 40KHZ crystal which is frequency multiplied to rates from 160 KHz to 16 MHz, in steps of 160 KHz. This frequency can be changed at any time under software control and will settle within about one millisecond, thus allowing you to dial up the exact amount of speed you need.

The SIM also includes a periodic interrupt timer which is clocked by a signal derived from the buffered crystal oscillator (EXTAL) input pin. The periodic timer can be set up as a real time clock by configuring it to generate a 1 second interrupt. Consult the Motorola manuals for further details.

PIC 16C64

The 16C64 is an EPROM based 8 bit CMOS microcontroller. One of the many features of this microcontroller is low current drain. The Model 8 is designed to use the PIC to wakeup from extended periods of inactivity while monitoring for the start of an event. In the lowest power mode (HYB), the PIC takes control of the Model 8 while the 68332 is stopped. The PIC can be programmed to wakeup the 68332 from HYB at fixed intervals and check the sensor inputs for measurements which would indicate need of the 68332.

Oscillator

The Model 8 uses a Harris HA7210; low power crystal oscillator with a Statek 40 KHz tuning fork quartz crystal. The crystal has an initial accuracy of $\pm 0.003\%$ at 25°C , and will operate from -40°C to $+85^{\circ}\text{C}$ with a stability of about $-0.035 \text{ ppm}/^{\circ}\text{C}^2$. This crystal also has excellent aging characteristics; however, like any crystal it will change over time.

For those applications requiring more accuracy, the Model 8 includes a provision for adding an external time base. To add an external time base, first disable the crystal oscillator by pulling pin 2 on the PR8 (CLKEN) low. The external time base can then be added on pin 58 of the PR-8 (40KHz).

Model 8 Accessories

I/O-8 Prototyping Board

The I/O-8 provides a place to build signal conditioning circuits. More importantly, the I/O-8 provides an easy place to connect the DC power source to.

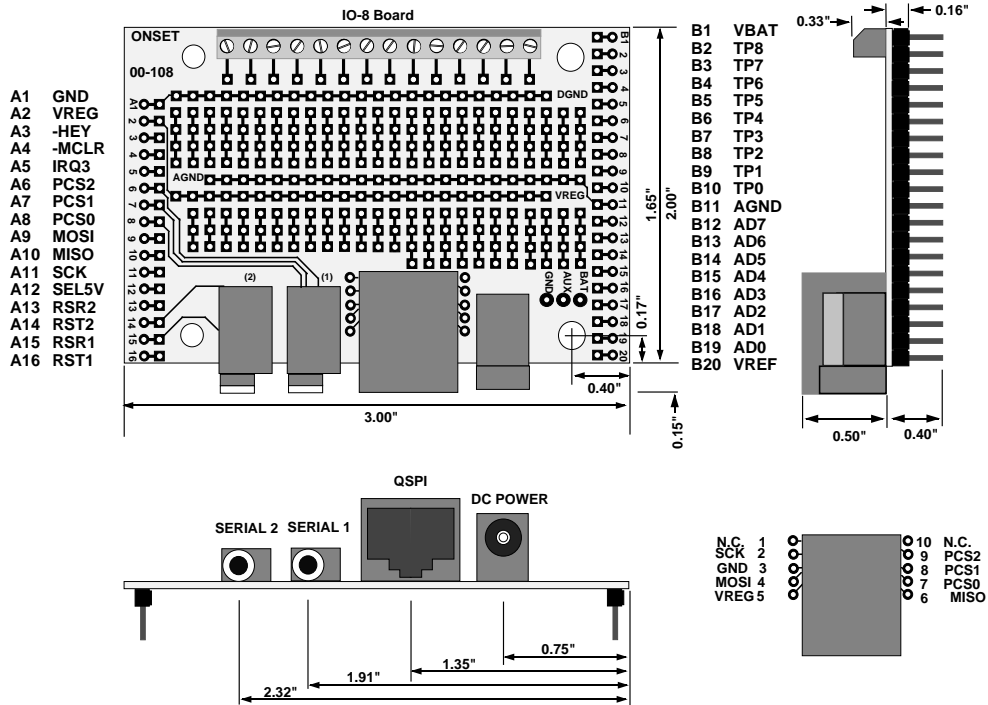


Figure 6-8: I/O-8 Prototyping Board

PR-8 Prototyping Board

The PR-8 has all the benefits of the I/O-8 prototyping board plus many more. This board connects to the Model 8 using two elastomeric connectors (SquishyBus) and gives control to the user for 110 pins as opposed to only 55 pins on the I/O-8 board.

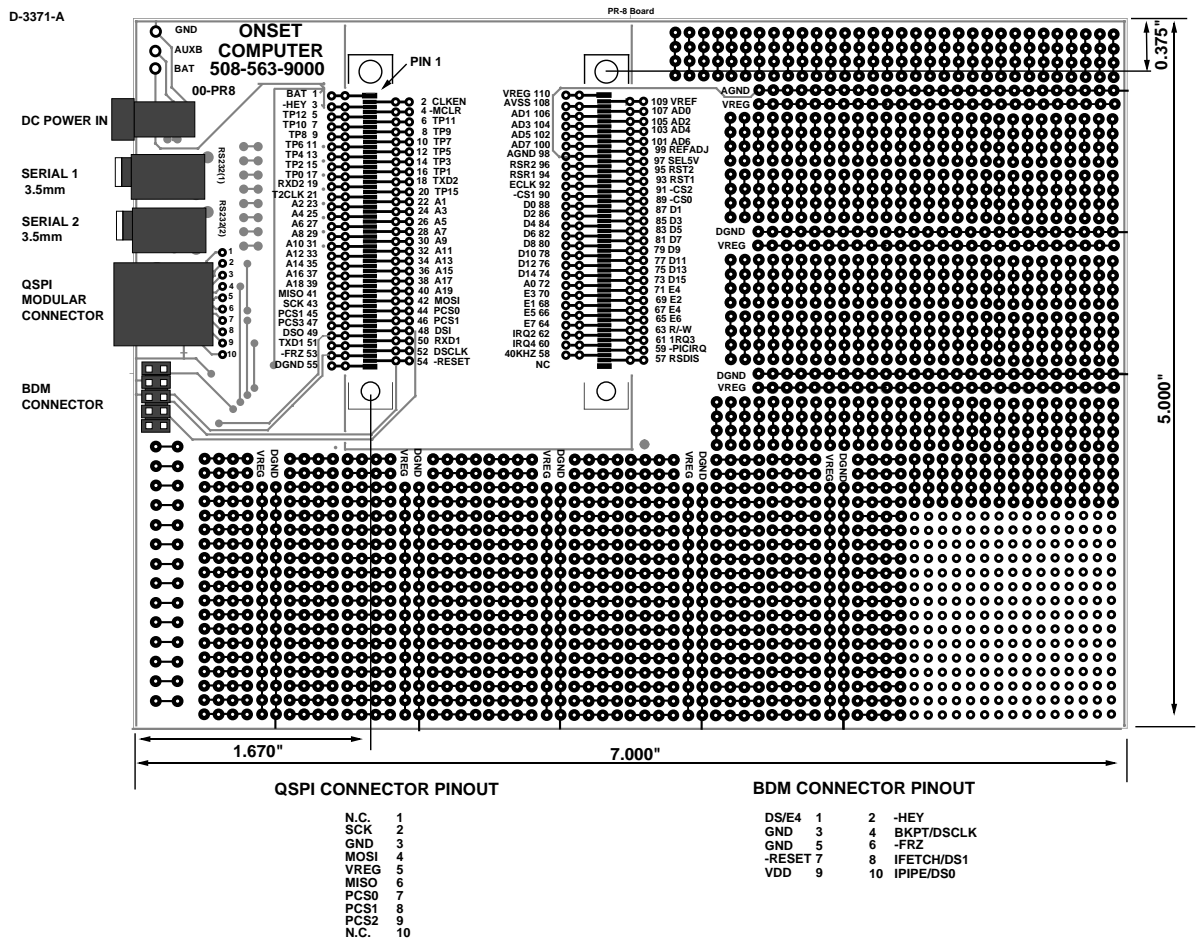


Figure 6-9: PR-8 Prototyping Board

Aztec C Compiler for DOS

This will allow you to write programs for the Model 8 and control the Model 8 in C language. If you plan on using any of the functionality of the BYOB (Build Your Own Basic), you will need a C compiler.

Aztec C Source Level Remote Debugger

This source level debugger will aid you in locating and correcting programming errors in C.

Tattletale 8 C Libraries for DOS

This package includes all the Model 8 specific commands that are used with the C language.

SquishyBus Connectors

These elastomeric connectors are used to connect the Tattletale Model 8 to the PR-8 prototyping board. These connectors feature a solderless connection that allows close spacing between pads, putting more contacts in a smaller area.

PCMCIA Card Adapter (Obsolete, contact Onset for alternate part.)

This adapter will allow you to purchase PCMCIA cards of whatever size is available on the market and be used with the Tattletale Model 8. We have tested individual software routines that will read and write the following cards throughout their address range. We will add more to this list as we test them.

Flash:

Intel Series 2+	20MB	P/N IMC020FLSP-15/20
Intel Series 2+	4MB	P/N IMC004FLSP-15/20
AMD	1MB	P/N AmC001BFLKA

SRAM:

FUJITSU	256K	P/N 90811-20
MITSUBISHI	2MB	P/N MF32MI-L6DAT-00

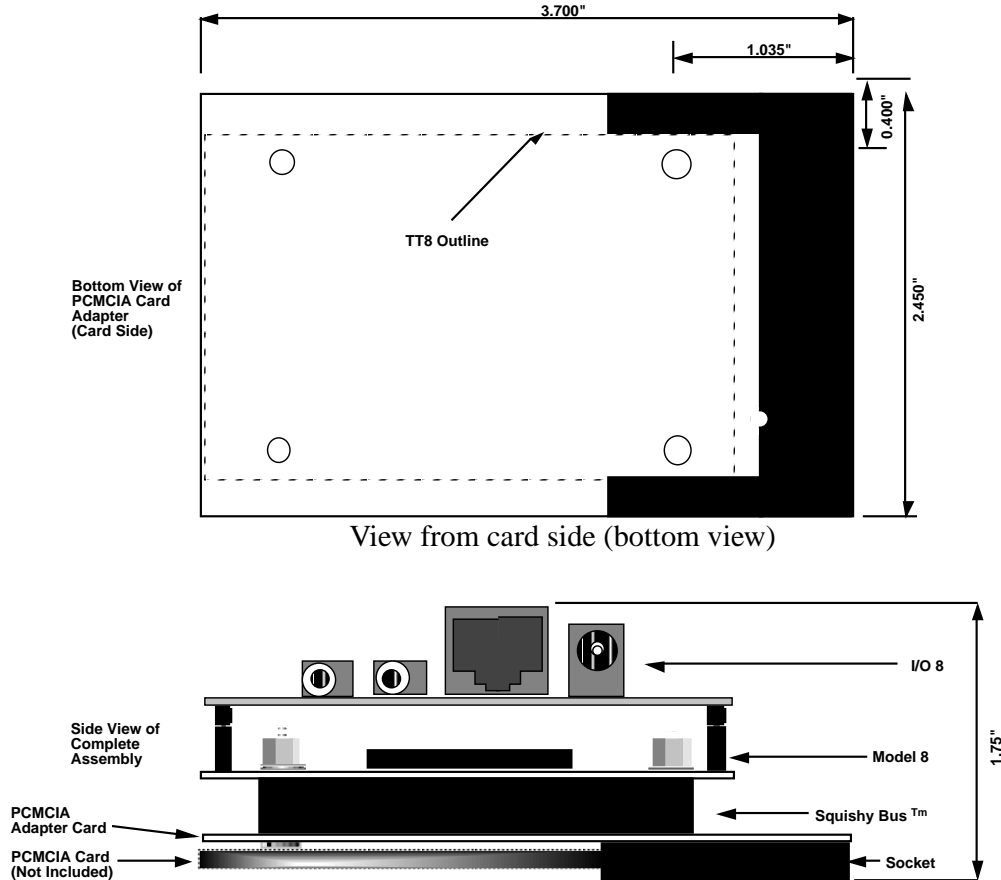


Figure 6-10: Side View of I/O8, TT8 and PCMCIA Adapter Assembly

Section 7 - Application Notes

Adding a 16 x 2 Display to the Tattletale

Adding a 16 x 2 character alphanumeric display to a Tattletale is very easy. With a one-chip interface to the SDO lines (D2, D5 and D11), you will be able to write to the display using SDO, giving you the full flexibility of the SDO command (The SDO data line on the Model 8 defaults to using D8).

The Interface:

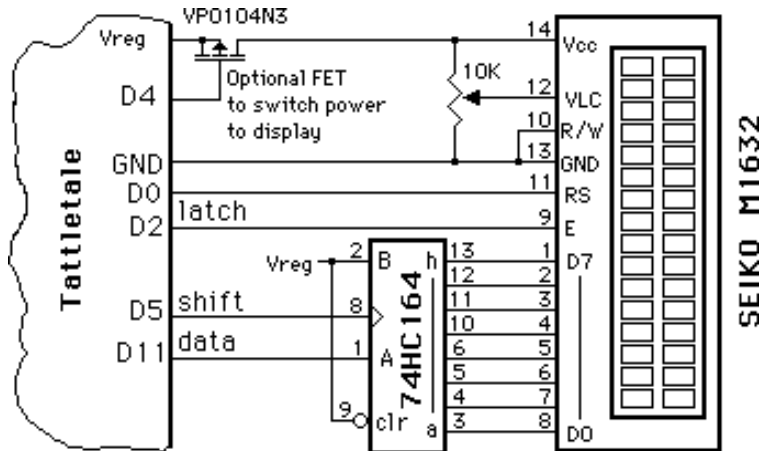


Figure 7-1: Adding a 16 x 2 Display to the Tattletale

In the oscillator-stopped low power modes (DONE, or HALT) of the Model 2B, 4A, 5 or 6, the I/O lines are tri-stated; all have pull-down resistors except D4 which has a pull-up. This makes D4 ideal for switching power to the display because its pull-up will turn the display off when the line is tri-stated. Send a zero to the display before powering it down so the lines coming from the 74HC164 will all be low.

The Display Manufacturer:

The Seiko M1632 is only one of many usable LCD displays. Hitachi and Seiko make a number of displays with identical interfaces.

Seiko: M1641 (16 x 1), M4032 (40 x 2) Hitachi: similar parts, different numbers

Software:

```
PCLR 0 // command mode
SDO 56,8
SDO 56,8 // set 8-bit data length twice
SDO "",\6,\12,\1; // increment pointer with each byte
// display on, no cursor, clear display
PSET 0 // data entry mode
SDO "HELLO FOLKS"; // some text, no CR, LF
A=TEMP(CHAN(7)) // now do second line
PCLR0
SDO &HC0,8
PSET0
SDO"TEMP= ",#1,A/10,".",A%10," C.";
```

The control commands are less than obvious; you will need the display manual to understand them fully. It's very easy to write even complex expressions to the display.

Storing Date and Time in Binary

This application note shows you how to store the date and time (with a resolution of a second) in the most compressed format. This assumes you set the ? variable with the correct real time by setting ?(5) = year, ?(4) = month, ?(3) = day, ?(2) = hour, ?(1) = minute, ?(0) = second and then used the STIME command to set the ? variable to this time. As long as power is maintained to the Tattletale, the ? variable will be updated every 0.01 seconds to keep the correct time on the Tattletale.

Whenever the RTIME command is used from now on, the current ? variable value is converted to year, month, day, hour minute and second and stored in the ? array. You could store these six values individually but this would take at least six bytes (if you're careful). This line will condense the current real-time to a four byte value:

```
RTIME : STIME A
```

Variable A will now contain the number of seconds since January 1, 1980. To convert the time back to six separate fields use:

```
RTIME A
```

and the variable ?(5) = year, ?(4) = month, ?(3) = day, ?(2) = hour, ?(1) = minute and ?(0) = second. You can use any Tattletale integer variable, we used A as an example.

This is also a great way to check for elapsed time between two events without worrying about "Thirty day months: September April, June and November. All the rest have 31 except February which might have 28 but then again...".

122 KHz A-D for Tattletale loggers (Model 4A only)

Add a Half-flash Converter

The MAX158ACP is a ADC820 half-flash converter with an 8-channel multiplexer ahead of it. With a conversion time of only 2.5 μ Sec and a particularly simple interface, it allows a Tattletale to make and store 8-bit burst measurements under assembly language control at an astonishing 122KHz (244KHz for a Model 2B or 6). We built it onto a PR-4 attached to a Model 4A to test it out.

It only draws about 6mA, but when connected as shown in Figure 7-2 it powers down to only about 35 μ A when in DONE or when the battery supply is below 6V. The only bad news is that you need to drive its analog inputs with a source impedance of less than 1K to keep from losing accuracy.

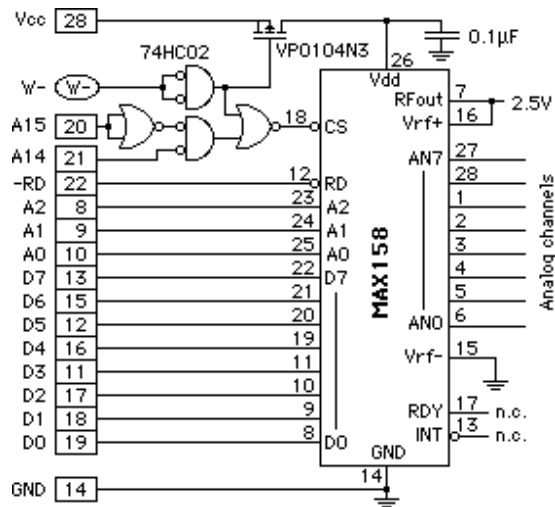


Figure 7-2: 122KHz A-D Circuit

All connections but W- are to the Model 4A's RAM socket. In DONE mode the 6303Y on the Model 4A tri-states all of its lines, but very little current comes from the address and data lines (which have 1 Meg pull-ups). By cutting off current to the chip's supply and driving -CS low we reduce the current drain to only a few microamps.

Example Program

By using the format function of the print command you can add a variable number of spaces. Each reading is the result of the previous conversion so the result of the first measurement is discarded. This simplified high speed example runs at a paltry 94K samples per second on a Model 4A or a 5, it would run at 188K samples per second on a Model 2B or 6. The 122KHz (244KHz for 2B or 6) example is on our bulletin board under the name:

FASTA-D.TXT (notice the dash between the A and D)

```

REM Make 256 measurements spaced by 10.5794µSec
ASM  &H118
      SEI
      XGDX
      CLRA
Loop LDAB &H8000
      STAB 0, X
      INX
      DECA
      BNE Loop
      CLI
      RTS
      END
CALL &H118,LABPTR (FINISHED)      REM write directly to the datafile
                                   REM Plot the results (5KHz sine wave)

X=1
FOR A=1 TO 255 :REM skip first measurement
ADValue=GET(X)
PRINT 'I',#ADValue/4,ADValue%4
NEXT A
STOP
FINISHED:

```

```

1      0
1      3
1      2
1      3
1      1
1      1
1      2
1      3
1      2
1      0
1      2
1      3
1      0
1      2
1      3
1      0
1      2
1      2
1      0

```


Adding More A-D channels

More analog inputs can be added to a Tattletale by using some of its digital lines to multiplex the analog inputs already on the board. Adding a 74HC4051 in front of one input as shown in Figure 7-3 multiplexes eight inputs into one input. Power the 4051 from the +V and ground lines so the multiplexer will be continuously powered. Three digital lines are lost to control the multiplexer.

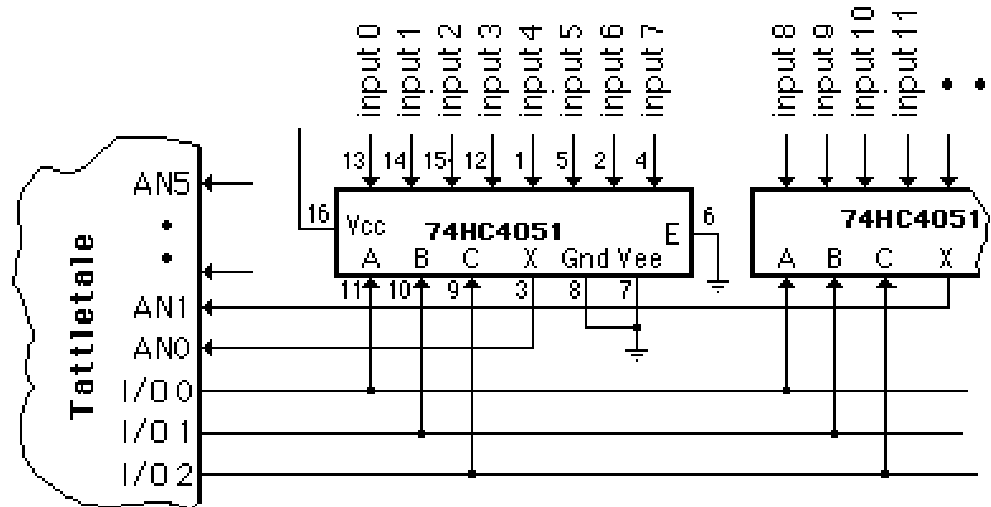


Figure 7-3: Circuit Diagram for Adding more A-D Channels

Before you can read a channel you must select the right input to the multiplexer by appropriately setting or clearing the I/O lines connected to the multiplexer.

```
PCLR 0,1,2
A=CHAN(0)           // this reads one channel
PCLR 1,2:PSET 0
A=CHAN(0)           // this reads another
```

You can even automate the channel selection.

```
DIM @(48)
FOR A=0 TO 47       // read all 48 channels to the @ array
  GOSUB label1     // returns channel 'A' result in X
  @(A)=X
NEXT A
STOP
label1: B=A
C=A%8
D=A/8              // 'C' = mux addr, 'D' = converter chan
FOR E=0 TO 2      // set the mux address one pin at a time
  IF B%2<>0 PSET E:GOTO label2 // set pin if bit is high
  PCLR E          // otherwise clear it
label2: B=B/2
NEXT E            // then do all three pins
X=CHAN(D)        // make conversion
RETURN           // return with result
```

12-Bit 10KHz A-D (Model 5 only)

This application note shows how to add a Linear Tech LTC-1290 8-channel of 12-bit A-D to a Model 5, giving you sample conversions at rates up to 10KHz. Note however that it connects to the main UART's UDI line. If your application can part with the Main UART's serial input while conversions are being made this will work just fine. You could control the A-D with a parallel port line instead of the UDI line, but the conversion would be much slower. Linear Tech gave us a sample LTC-1290 and our sample drew 300µA in its 'POWER SHUTDOWN' mode and 4mA while running.

The channel input is selected by A0, A1 and A2, strobed in by 'STROBE'. A separate shift register was needed since the 6303 synchronous mode is not full duplex. The shift register is set to select single ended, unipolar, LSB first, and 16-bit word length. The 10K resistor is needed to isolate the TC-4 from the LTC1290. Note that the conversion is ratiometric to the LTC-1290's 5V supply.

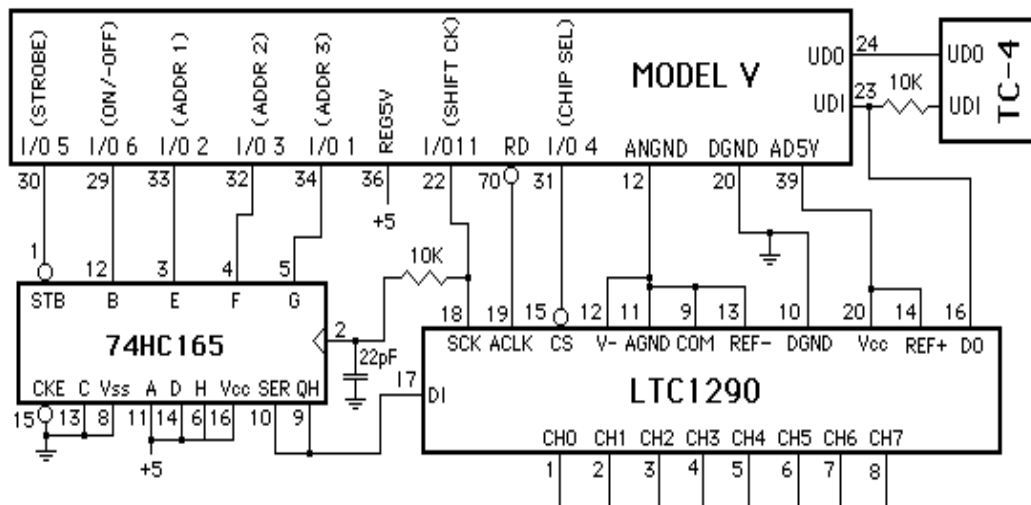


Figure 7-4: 12-Bit 10KHz A-D Circuit for the Model 5

```

// Channel 0, Power on and latch it
GOSUB label2 // RUN ASSEMBLER, LOAD INTO MEM.
PSET 11 : PCLR 1,2,3 : PSET 4,6 : PCLR 5 : PSET 5

label1: CALL Setup,0 // SET UP UART FOR USE BY A/D CONV
        CALL Conv,0,R // MAKE A CONVERSION
        CALL RUART,0 // SET UART TO NORMAL STATE
        PRINT R // PRINT A/D READING
        SLEEP 0:SLEEP 2 // TIME FOR UART TO FLUSH
        GOTO label1

// ROUTINE TO SET UP UART FOR USE BY A/D CONVERTER
label2: ASM &H74C0
Setup CLR &H11 ; CLEAR BITS 1,3 OF 11H TO DISABLE UART
      LDAA #&H10
      STAA &H10 ; 8-BIT SYNCHRONOUS, E/2 CLOCK

```

```

OIM 8,&H11 ; RESTART UART IN RECEIVE CONFIGURATION
RTS

; ROUTINE FOR STARTING CONVERSION (ADDR IN 74HC165)
; RETURN PREVIOUS RESULT IN A/B REGISTERS - TAKES 47.2 µSecs

Conv    AIM &HF7,&H15
        LDAA &H11
        LDAA &H12 ; CS LOW, SEND COMMAND WORD
        CPX 0
        CPX 0
        CPX 0
        CPX 0 ; 16 E CYCLE DELAY FOR TRANSMIT
        LDAB &H11
        LDAB &H12 ; LSB RECEIVED
        CPX 0
        CPX 0
        CPX 0
        CPX 0 ; 16 E CYCLE DELAY FOR TRANSMIT
        LDAA &H11
        LDAA &H12
        OIM 8,&H15 ; CS HIGH
        RTS

; MAKE SURE NOT TO RE-ENTER THIS ROUTINE
; FOR AT LEAST 58 READ PULSES (ALLOWS NEXT
; CONVERSION)

RUART   ; RESTORE UART TO NORMAL STATUS (SEND AND RECEIVE AT 9600 BAUD)
        CLR &H11 ; DISABLE UART
        LDAA #5
        STAA &H10 ; BAUD RATE FROM TIMER1
        OIM &H42, &H1B ; ENABLE TIMER2 INTERRUPTS AND E/128 CLOCK
        LDAA #95
        STAA &H1C ; TIMER COUNT TO GET 100 HZ RATE
        OIM &H1A,&H11 ; RESTART UART
        RTS
        END
        RETURN

```

Other Protocols for the Tattletale UART

NOTE: This application note can NOT be used with the Model 8.

Power up Default

On power up all Tattletaes default to 8 data bits, 1 stop bit, and no parity. The UART on all of the Tattletaes are capable of 11 other protocols (as well as four different baud rates). Changing the baud rate is covered in the manual, but changing protocols is not.

The UART Ports

To change the UART protocol you will need to modify two of the UART's registers. The relevant bits are shown in Figure 7-5.

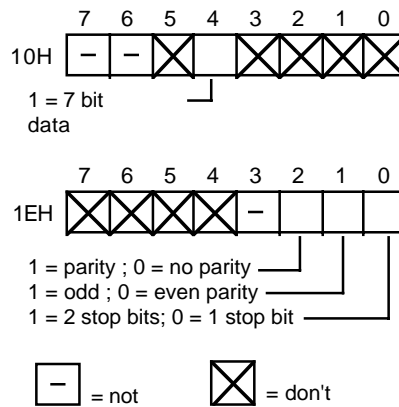


Figure 7-5: Modifying the UART Registers

Table 7-1: Possible Tattletale Protocol Settings

Protocol Option	Protocol Settings
1	8 data bits, 1 stop bit, no parity (the Tattletale default)
2	8 data bits, 1 stop bit, even parity
3	8 data bits, 1 stop bit, odd parity
4	8 data bits, 2 stop bits, no parity (or 'marking parity' because parity is always 1)
5	8 data bits, 2 stop bits, even parity
6	8 data bits, 2 stop bits, odd parity
7	7 data bits, 1 stop bit, no parity
8	7 data bits, 1 stop bit, even parity
9	7 data bits, 1 stop bit, odd parity
10	7 data bits, 2 stop bits, no parity (or 'marking parity' because parity is always 1)
11	7 data bits, 2 stop bits, even parity
12	7 data bits, 2 stop bits, odd parity

The protocol information is stored at two addresses: 10H and 1EH. Depending on the change you need, you may have to change one or both addresses.

Example Program

The sample program that follows shows two lines for each of the four possible choices. Choose the one that matches your needs and ignore the other. When assembled, this routine takes up 19 bytes (&H118 through &H12A).

NOTE: Don't try to shorten the procedure, we have found that all of the steps are needed.

```

GOSUB label1           //Assemble the routine in the lines below
CALL Protocol,0       //Execute the assembly routine
<the rest of your program>
STOP
'

label1:  ASM &H118
Protocol AIM &HF5, &H11           ; Disable UART by clearing bits 1 and 3
        OIM &H10,&H10           ; 7 data bits
        (or AIM &HEF,&H10)       ; 8 data bits
        OIM 1,&H1E             ; 2 stop bits
        (or AIM &HFE,&H1E)       ; 1 stop bit
        OIM 4,&H1E             ; Parity enabled
        (or AIM &HFB,&H1E)       ; Parity disabled
        OIM 2,&H1E             ; Odd parity (if enabled)
        (or AIM &HFD,&H1E)       ; Even parity (if enabled)
        OIM &HA, &H11           ; Enable UART again by setting bits 1 and 3
        RTS                   ; Return from assembly subroutine
        END
        RETURN                 ; Return from TxBASIC subroutine

```

Convert Bipolar Input to Unipolar

Signals that go both Positive and Negative

We have been asked how to interface a bipolar signal to a Tattletale. Some Tattletalets (like the Model 4A and 6) can be made to run in bipolar mode (make sure to add a Schottky diode between V₋ and ground), others won't. This note shows a simple way to convert a bipolar input to a positive only signal.

The simple schematic in Figure 7-6 shows an operational amplifier connected in an inverting configuration. Note that this design assumes a relatively low impedance signal is driving V_{in}.

The only tricky thing we have done is to give a positive bias to the non-inverting input to the amplifier. Remembering that op-amps adjust V_{out} so that the inverting and non-inverting inputs have the same voltage. The equation for this configuration is:

$$V_{set} = V_{in} + (V_{out} - V_{in}) * R1 / (R1+R2),$$

or, rearranged a little: $V_{out} = ((R1+R2) * (V_s - V_{in}) / R1) + R1$

You can change this to two equations, one for gain, another for offset:

$$\text{Gain} = \Delta V_{out} / \Delta V_{in} = - R2 / R1, \quad \text{and offset (V}_{out} \text{ for } V_{in} = 0) \quad \text{Offset} = V_{set} * (R1 + R2) / R1.$$

$$\text{or } R2 / R1 = - \text{Gain}, \quad \text{and } V_{set} = \text{Offset} * R1 / (R1+R2)$$

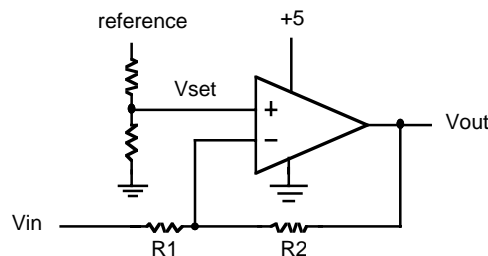


Figure 7-6: Circuit for Converting Bipolar to Unipolar

Example Application:

Suppose we had a signal that ranged from -0.5V to $+0.5\text{V}$, and wanted to translate that to a voltage we could read with our positive signal only A-D. We first add a 2.5V reference to the A-D and translate our input to a signal ranging from 0 to 2.5V . For this we would need a gain of 2.5 , and an offset of 1.25V . Our circuit actually gives a negative gain so that -0.5V would translate to 2.5V , and $+0.5\text{V}$ will translate to 0V . From the gain and offset equations we can find the ratio $R2/R1$ (2.5) and V_{set} (0.357V). If we use standard 1% resistors and the op-amp from Linear Technology we get the schematic shown in Figure 7-7.

We show a 2.5V reference, also from Linear Tech, fed by a 3K resistor and divided down to reach the 0.357V. The resistors are not the exact values that we need to give the gain and offset we wished for, but come very close. The LT1077's input offset current is 0.25nA max., giving an offset of about 25 μ V at the input. This, combined with the input offset voltage of 40 μ V max. is still less than 1 LSB of a 12-bit converter.

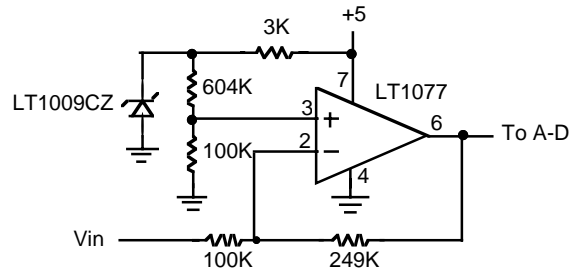


Figure 7-7: Another Circuit for Converting Bipolar to Unipolar

Operational Amplifiers and Instrumentation Amplifiers

For those of you that are not familiar with the terms “operational amplifier” and “instrumentation amplifier”, this very brief explanation should be enough to get you started.

Operational Amplifiers

The op amp (as they are commonly called) is an amplifier with both an inverting and non-inverting input, and a single output. For our purposes we will treat it as having an infinite gain, and no current flows through either input. Two commonly used circuits for op amps are shown in Figure 7-8, with the '-' designating the inverting and the '+' designating the non-inverting inputs.

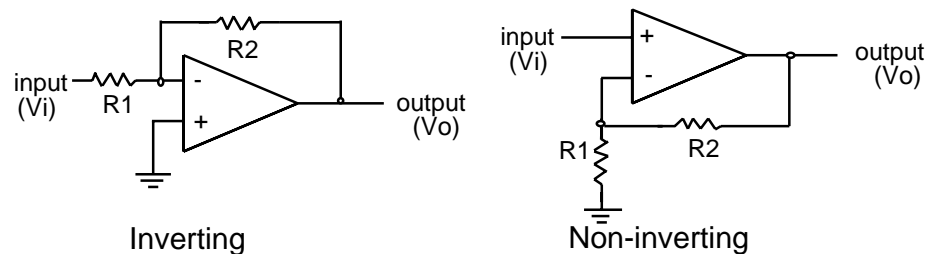


Figure 7-8: Operational Amplifiers

The Inverting Configuration

The inverting circuit has a gain of $-R2/R1$. This makes sense since the op amp will do what is needed to keep the + and - inputs at the same voltage, and any current flowing into the input resistor (R1) will also flow through the 'feedback resistor' (R2). An input voltage V_i will cause a current $V_i/R1$, and cause an output voltage V_o of $-R2(V_i/R1)$. Note the minus sign: the current flows through R2 away from ground if the current through R1 is flowing toward ground.

An op amp is not magic and it cannot give an output that is larger than its positive supply or lower than its negative supply. The Tattletale has a regulated 5V, but its negative supply is ground, so most applications will supply the op amp with ground and +5V. This makes it impossible to use the inverting circuit as drawn, unless the input is a negative voltage.

Advantages:

R1 protects input from external voltage

Disadvantages:

To get a large gain, R1 must be small, loading the input.

Requires negative input (in Tattletale application, without adding a negative supply)

The Non-inverting Configuration

Following the same analysis we did for the inverting configuration, we find that the gain of the circuit is $(R1+R2)/R1$ for the non-inverting circuit. In this case we are limited to positive inputs, since the Tattletale has no negative supply.

Advantages:

High input impedance (will not load down your signal source)

Disadvantages:

Requires positive input (in Tattletale application, without adding a negative supply)

Op amps are not perfect. Some limitations are described in Table 7-2.

Table 7-2: Op Amp Limitation Data

Item	Limitation
Current drain:	In your Tattletale application you won't want to use any more power than needed. Check the current drain specs on the part you choose.
Supply voltage:	Not all op amps work with supply voltages of +5 and ground. In some applications you may even want your circuit to work while the Tattletale is running from 3 volts.
Max output swing:	Most op amps can only drive to within about a volt of their positive supply, others to only about a volt of their negative supply.
Input voltage range:	Not all op amps are pleased with inputs close to the supply lines; some will work with input voltages <u>below</u> ground. Don't be confused by the inverting configuration described above. In that circuit the input voltage is always at ground.

Table 7-2: Op Amp Limitation Data

Item	Limitation
Input offset voltage:	The input offset voltage can be understood as the voltage difference between the positive and negative inputs needed to make the output voltage zero. Ideally this should be zero; in reality many op amps have input offsets of 10mV or more. This offset will appear as an error at the output equal to the input offset times the gain of the circuit.
Input offset current:	Many op amps have FET inputs that take almost no current at the input, but some can have sizable currents flowing (nanoamps) into or out of the inputs. This can lead to sizable errors if your input resistor is a large value.

That's not all, but should be enough to give you some idea of what to look for in op amp data sheets.

The Op Amps we use:

We will show some of our biases by giving you short descriptions of three op amps we like. Note that each has its own strengths.

LT1077, 1078, 1079

Single, dual and quad amplifier from Linear Technology (408) 432-1900. This part has lovely current drain, input voltage range and input offset voltage specs.

Current drain:	40 μ A typical for each amplifier (μ A = microamp, a millionth of an amp)
Supply voltage:	3 volts minimum
Max output swing:	ground +6mV to positive supply -1 Volt (mV= millivolt, 1/1000V)
Input voltage range:	ground -5V to positive supply
Input offset voltage:	30 μ V typical (μ V = microvolt, a millionth of a volt)
Input offset current:	0.25nA max. (nA = nanoamp = a milli-micro amp)

TLC1078

Dual amplifier from Texas Instruments (800) 232-3200. Note the superb input offset current specifications.

Current drain:	15 μ A typical for each amplifier
Supply voltage:	1.4V minimum
Max output swing:	ground to positive supply -1 Volt
Input voltage range:	ground -0.2V to positive supply -1 Volt
Input offset voltage:	160 μ V typical
Input offset current:	0.1pA max. (pA = picoamp = a micro micro amp)

ALD1704, 1701, 2701, 4701

Single, dual and quad amplifier from Advanced Linear Devices (408) 720-8737. This part drives line to line.

Current drain:	120 μ A typical for each amplifier
Supply voltage:	2.0V minimum
Max output swing:	ground to positive supply -1 Volt
Input voltage range:	ground -0.2V to positive supply -1 Volt
Input offset voltage:	2mV <u>for the premium version</u>
Input offset current:	25pA max. (pA = picoamp = a micro micro amp)

NOTE: If you need the specifications for gain-bandwidth product, noise characteristics, stability, offset drift, temperature coefficients or output drive characteristics, refer to the data sheets from the manufactures. We do not supply them in this manual. These and other parameters are important in some applications, but we don't have the space to cover everything.

Instrumentation Amplifiers

What if your sensor provides two outputs (like a bridge circuit)? An operational amplifier has both an inverting and a non-inverting input, but one is needed to set the gain of the circuit. You can build an instrumentation amplifier from two operational amplifiers and a small pile of precision resistors using the circuit below:

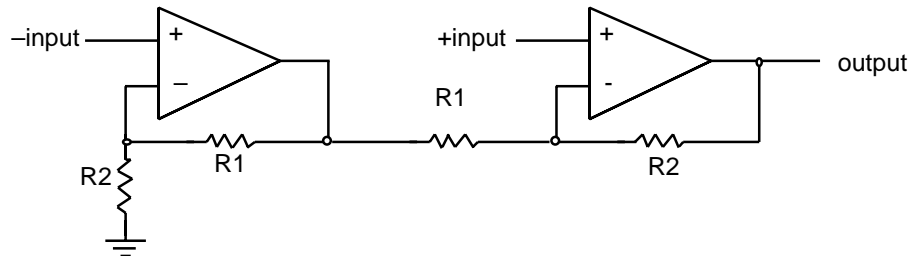


Figure 7-9: Instrumentation Amplifier Circuit

This circuit has a gain $G = (R1+R2)/R1$, and works over a range of input voltages that go from ground to 'Max positive output' * $R2/(R1+R2)$.

LT1101 Instrumentation Amplifier:

Linear Technology's LT1101 packages this all in one 8-pin part and can be set to a gain of 10 or 100 with no external parts.

Current drain:	75 μ A typical
Supply voltage:	2V minimum
Max output swing:	ground +4mV to positive supply -1V
Input voltage range:	ground +70mV to positive supply -2V
Input offset voltage:	50 μ V typical
Input offset current:	8nA max.

Digital Output Protection

Digital outputs are just as vulnerable as inputs, and they cannot be driven, even transiently, with a signal larger than the 'V' supply or lower than ground. They also have relatively low drive capability and should be buffered if they are expected to drive substantial loads. Some examples of output buffering are given in Figure 7-10.

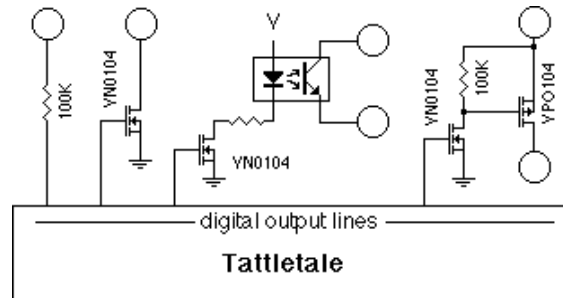


Figure 7-10: Digital Output Protection Circuit

- 100K resistor** The resistor in the first example will provide protection but very little drive current. It is suitable for connecting to CMOS inputs driven from a separate supply (where there is a possibility that the supplies won't track).
- VFET driver** The VN0104 is an N-channel VMOS transistor with an on-resistance of about 4 ohms when the gate is +5V and lots of megohms when the gate is grounded. In the configuration shown, it is suitable for driving small relays or opto-isolators. The VN0104 has a V_{ds} of 40V, so the solenoid can be powered directly from Vbat or any other convenient source. The VN0104 and VP0104 parts are available from Supertex, 1225 Bordeaux Dr., Sunnyvale CA 94088-3607, phone (408) 744-0100.
- Power switch** The last example shows how to use two VFETs (one P-channel and one N-channel) to, for example, turn on and off a separate voltage to enable Vbat to a separate regulator.

Digital Input Protection

The digital inputs should be protected from signals that exceed the Tattletale's internal bus supply levels. Figure 7-11 shows five techniques that protect the inputs from large current spikes which may cause latch-up.

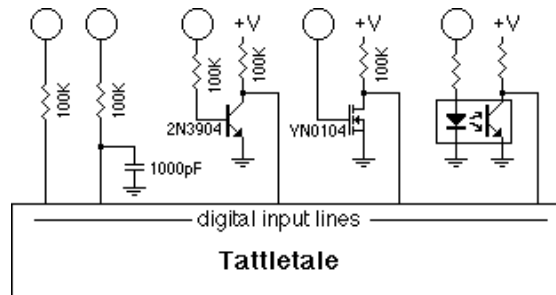


Figure 7-11: Digital Input Protection Circuit

- | | |
|-----------------------|--|
| Resistor | The resistor protects the input from surges by limiting the amount of current that can be injected. |
| R - C | Adding a capacitor helps protect inputs from high voltage spikes caused by electrostatic discharge. |
| Transistor | This solution isolates the Tattletale's inputs completely from the source, placing the transistor at hazard (the transistor is a lot easier to replace than the 64-pin microprocessor that the inputs are connected to). |
| VFET | Same as the transistor, but takes less current. |
| Opto-isolation | This solution has the advantage of total isolation of both the supply and the ground of the source from that of the microprocessor. However, it comes at the expense of a substantial current drain. |

Input Protection

If sensors are mounted external to the Tattletale, and particularly if they are made so that they can be disconnected, you should provide some protection so that static discharge will not damage the Tattletale's A-D converter or your signal conditioning circuitry. Some input protection circuits are shown in Figure 7-12, Figure 7-13 and Figure 7-14.

Using just a Resistor for Protection

This is minimal protection for a line that goes out of the Tattletale box. For A-D converter inputs the resistor (plus impedance of the source) should not be larger than about 3K since the converter has a capacitor that must fill to 12-bit accuracy during its sampling time. Digital inputs can have values of 100K; the only limitation is input leakage and stray leakage on the board. Op amps will see an input offset equal to the product of the resistance and the input offset current.

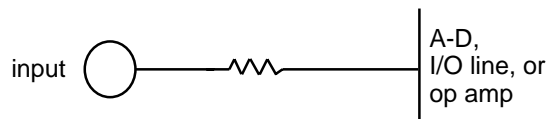


Figure 7-12: Input Protection with just a Resistor

Using a Resistor and a Capacitor for Protection

Figure 7-13 is slightly more complex, and helps protect the input from momentary voltage transients such as mild static discharge, but can significantly distort AC signals at the input. Make sure that the time constant of the circuit ($t = RC$) is short compared to the expected change rate of your input signal. Typical resistor and capacitor values of 100K and 0.1 μ F are fine for digital lines but analog lines must have very low resistor and capacitor values. Analog line resistors (plus source impedance) should never go above 3K and capacitors shouldn't go above 0.001 μ F.

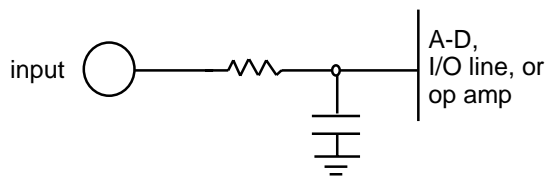


Figure 7-13: Input Protection with a Resistor and Capacitor

Using a Resistor, a Capacitor and a Zener Diode for Protection

Add a 6.3V zener diode and you will have very good input protection. The zener makes sure your input never gets far outside the supply lines, and the resistor limits the current into the zener. See the text above for the analog input lines. For the digital I/O lines, a 100K resistor is not inappropriate for both resistors. This circuit is very effective against static discharge, but is not adequate protection against lightning.

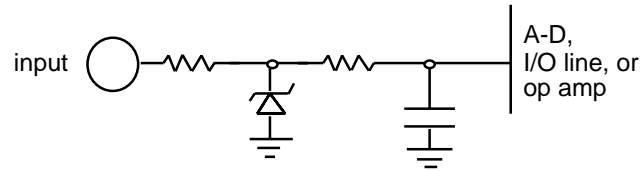


Figure 7-14: Input Protection with a Zener Diode



NOTICE



Lightning! - If you connect your Tattletale to a long cable somewhere outside, lightning damage is a possibility. The strike does not need to hit the wire directly, just near enough to induce a large voltage transient. To protect against lightning, place a spark gap at the input. Since there is no room in the Tattletale for the spark gap, you would have to mount it on a terminal block external to the Tattletale.

Alternate System Clock

NOTE: This application note can NOT be used with the Model 8.

This application note explains a bit about the alternate system clock in TxBASIC, why you might want to use it and how to use it. Refer to the two programs included in this application note (ALTCLOCK.TXB and NEWBAUDS.TXB).

Overview of the Timing Sub-system of a Tattletale

The microprocessor used in the Tattletale has two internal timers. Timer1 is the more flexible of the two and can be used for measuring the time between events or producing an output on a timed event. Timer2 is better used to produce an event at a fixed time interval. In TxBASIC, we use Timer2 as the system clock. This simply has to produce an interrupt at a fixed interval (nominally 0.01 sec). Timer1 is used by the PERIOD and COUNT commands to measure the time between cycles of signals with varying frequencies. Timer1 is also used by the TONE and DTOA commands to produce signals with a variable frequency.

The microprocessor used in the Tattletale also includes a UART for serial communications. In the default mode, the UART is capable of only 4 different baud rates (and only 3 of these are really useful). And the baud rate depends on the speed of the crystal on the Tattletale. A 4.9 MHz crystal allows only baud rates of 300, 1200, 9600 and 76800. A crystal that is 9.8 MHz allows only baud rates of 600, 2400, 19200 and 153600.

Need for an Alternate System Clock

We can add more flexibility to the UART because Timer2 has a mode allowing it to be used as the baud rate generator for the UART. This allows any of these baud rates: 300, 600, 1200, 2400, 4800, 9600, 19200 and 38400 with either the fast or slow crystal. This is great except that it leaves us with no system clock. This means that neither the time (the ? variable) nor the SLEEP counters will be updated. Also, any commands that use a time-out (like OFFLD, UGET, ITEXT and XSHAKE) will not work.

This is where the alternate system clock comes in. We can use Timer1 to produce a periodic interrupt and then direct the interrupt to a section of code that will mimic the system clock. You cannot do this on versions of TxBASIC before version 2.08.

Using the Alternate System Clock

Before using the new alternate system clock, you must be sure that you will not use the PERIOD, COUNT, TONE or DTOA commands while it is in use. We may add a lock-out feature in a future release of TxBASIC to enforce this but for now we don't.

First, you must convert the system timer from its default mode to the alternate system clock. Use the subroutine in ALTCLOCK.TXB called 'alternateClock'. You can then change the baud rate using the 'UARTnewbaud' in the program NEWBAUDS.TXB. After you are done with the UART, you can restore it to its default condition with the 'UARTdefault' subroutine in the NEWBAUDS.TXB program. Finally, you would want to restore the system clock to its default state using the 'standardClock' subroutine in ALTCLOCK.TXB program.

NOTE: The program ALTCLOCK.TXB is a stand-alone program but the NEWBAUDS.TXB program contains just subroutines that can be copied into the program you want.

There are two lines that must be changed to match your situation. If your Tattletale has a slow crystal (as in the 2A, 2A-32, 5 and 5F-LCD), change the line "add #F_XLT" to "add #S_XLT" (in the 'alternateClock' routine).

In the 'UARTnewbaud' routine, change the line "ldaa #FB_300" to use the constant for your crystal speed and baud rate. That's it.

We've designed the alternate system clock so it will work correctly with the RATE command, too. If you use RATE at any time, it changes the prescale values for both the standard system clock and the alternate system clock so you don't need to use RATE each time you change system clocks.

NOTE: You will find that each time you switch the system clock, you may lose a fraction of a time tick. Don't switch too often until we've had more time to analyze this and figure out a cleaner switch method.

In the ALTCLOCK.TXB program, the line that reads: "ldd #H'FFC1; address of ALTCLK interrupt handler" is an assembly language subroutine provided by newer versions of TxBASIC that knows how to handle TIMER1 and the system clock.

Alternate Clock Program (ALTCLOCK.TXB)

```
// This sample program will set up Timer1 to be used as the system clock,
// then disable Timer2 from being the system clock and finally enable
// Timer1 to start being the system clock. That is all done by subroutine
// 'alternateClock'. Then the time is printed out once a second to show
// that the ? variable is being updated. This will continue until you press
// a Ctrl-C which vectors to the 'exit' label and the 'standardClock'
// subroutine is called to disable Timer1 from being the system clock and
// to enable Timer2 to again be the system clock
```

```
    cbreak exit          // jump to label 'exit' when Ctrl-C is hit
    gosub alternateClock // use Timer1 as system clock
    sleep 0

loop:                                // loop forever until you hit Ctrl-C
    rtime                      // convert ? variable to real time in ? array
    print #02,?(2),“:”,?(1),“:”,?(0),\13;
    sleep 100
    goto loop

exit:                                // get here when Ctrl-C is hit
    print
    print "Restoring Timer2 as system clock"
    gosub standardClock // go back to using Timer2 as system clock
    stop
```



```

//***** Subroutines used by example program *****

```

```

// This subroutine stops Timer2 from being system clock and enables
// Timer1 to be the system clock. If your Tattletale has a slow crystal,
// change the line "add #F_XTL" to "add #S_XTL".

```

```

alternateClock:

```

```

asm $
TIME    equ  H'43    ; address of LS byte of ? variable
FRC     equ  H'09    ; address of Free Running Counter
OCR1    equ  H'0B    ; address of Output Compare Register 1
ICR     equ  H'0D    ; address of Input Compare Register
OCI     equ  H'106   ; Output Compare Interrupt vector address
TCSR1   equ  H'08    ; address of Timer Control Status Register 1
TCSR2   equ  H'0F    ; address of Timer Control Status Register 2
TCSR3   equ  H'1B    ; address of Timer Control Status Register 3
TCR     equ  H'1C    ; address of Time Constant Register (Timer2)
TCRCPY  equ  H'94    ; copy of the Timer2 Time Constant Register (TxBASIC)
F_XTL   equ  D'24576 ; number Timer1 counts per 0.01 sec (fast crystal)
S_XTL   equ  D'12288 ; number Timer1 counts per 0.01 sec (slow crystal)

wait1   ldaa  TIME    ; get LS byte of current time (updated every .01 sec)
        slp                    ; wait for next interrupt
        cmpa  TIME    ; if interrupt was clock, this will have changed
        beq  wait1    ; branch if not different (not a clock tick)
        ldd  FRC     ; get current Free Running Counter value
        add  #F_XTL   ; add number of FRC ticks / 0.01 sec
        std  OCR1    ; value of FRC for next clock tick
                        ; (handled automatically after first tick)
        aim  #H'BC,TCSR3 ; disable Timer 2 interrupt
        ldaa #H'7E    ; 'JMP' opcode
        staa OCI     ; store at OCI vector
        ldd  #H'FFC1 ; address of ALTCLK interrupt handler
        std  OCI+1   ; address to 'JMP' to
        clra
        staa TCSR2   ; disable other Timer 1 interrupts and output lines
        ldaa #H'08    ; 'interrupt on Timer 1 compare' bit
        staa TCSR1   ; allow Timer 1 interrupts to act as clock
        end
        return

```

```

// this subroutine returns to the normal Timer2 controlled system clock

standardClock:
asm $

wait2    ldaa    TIME    ; get LS byte of current time (updated every .01 sec)
         slp      ; wait for next interrupt
         cmpa    TIME    ; if interrupt was clock, this will have changed
         beq     wait2   ; branch if not different (not a clock tick)
         ldaa    #H'02
         staa    TCSR1   ; stop Timer 1 interrupts
         ldaa    TCRCPY  ; get original value of Timer2's Time Constant Register
         ldaa    TCR     ; and restore it
         ldaa    #H'52
         staa    TCSR3   ; enable Timer 2 interrupts to act as clock
         end
         return

```

New Baud Rate Program (NEWBAUDS.TXB)

```

// These subroutines can be used to get many different baud rates from the
// Tattletale's main UART but you should first set up the alternate system
// clock. Refer to the Alternate System Clock text and the sample program
// ALTLOCK.TXB for additional information.

```

```

// The first subroutine causes Timer2 to be used as the baud rate generator
// for the UART and allows you to pick one of the standard baud rates. In
// this sample, we use the baud code assuming we have a fast crystal and
// want a baud rate of 300. You can choose the constant you want.
// The second subroutine returns the UART to its default condition.

```

```

// Subroutine #1 to allow any baud rate

```

```

UARTnewbaud:
asm $

TCR      equ H'1C    ; address of Timer2 Time Constant Register
TCRCPY   equ H'94    ; copy of TCR kept by TxBASIC (TCR is write only)
TCSR3    equ H'1B    ; address of Timer Control/Status Register 3
RMCR     equ H'10    ; address of Rate/Mode Control Register (for UART)
TRCSR1   equ H'11    ; address of Tx/Rx Control Status Register 1 (for UART)

FB_38400 equ 1       ; baud code for fast crystal, 38400 baud
FB_19200 equ 3       ; baud code for fast crystal, 19200 baud
FB_9600  equ 7       ; baud code for fast crystal, 9600 baud
FB_4800  equ 15      ; baud code for fast crystal, 4800 baud
FB_2400  equ 31      ; baud code for fast crystal, 2400 baud
FB_1200  equ 63      ; baud code for fast crystal, 1200 baud
FB_600   equ 127     ; baud code for fast crystal, 600 baud
FB_300   equ 255     ; baud code for fast crystal, 300 baud

```

```

SB_38400 equ 0 ; baud code for slow crystal, 38400 baud
SB_19200 equ 1 ; baud code for slow crystal, 19200 baud
SB_9600 equ 3 ; baud code for slow crystal, 9600 baud
SB_4800 equ 7 ; baud code for slow crystal, 4800 baud
SB_2400 equ 15 ; baud code for slow crystal, 2400 baud
SB_1200 equ 31 ; baud code for slow crystal, 1200 baud
SB_600 equ 63 ; baud code for slow crystal, 600 baud
SB_300 equ 127 ; baud code for slow crystal, 300 baud

    aim &HF5,TRCSR1 ; disable UART while changing its setup
    oim &H20,RMCR ; select Timer2 as baud rate generator
    aim &HBC,TCSR3 ; disable Timer2 from causing interrupt, use E clk
    ldaa #FB_300 ; load baud code for: fast crystal, 300 baud
    staa TCR ; store in Time Constant Register (Timer2)
    oim &HA,TRCSR1 ; enable UART
    end
return

// Subroutine #2 to reset UART back to its default state

UARTdefault:
asm $
    aim &HF5,TRCSR1 ; disable UART while changing setup
    aim &HDF,RMCR ; baud rate from crystal
    oim &H52,TCSR3 ; enable Timer2 interrupts and E/128 clock
    ldaa TCRCPY ; get original value of Time Constant Register
    staa TCR ; restore it
    oim &HA,TRCSR1 ; restart UART
    end
return

```


Section 8 - Troubleshooting

Introduction

This section will help you diagnose problems with the Tattletale and either correct them or instruct you to call Onset Computer Product Support for additional help.

NOTE: If you use the Tattletale Model 8 with the Aztec C compiler and you need support for problems with Aztec C, contact the Technical support at Aztec C. The Model 8 is the only Tattletale that can use the Aztec C compiler.

Troubleshooting Tattletale Problems

Use the flow chart in Figure 8-1 to help determine the problem with the Tattletale. Refer to Table 8-1 while performing any of the test procedures.

Table 8-1: Tattletale Communication Default Settings

Tattletale Model	Protocol	Setting
2A, 3, 4, 4A, 5, 5F-LCD and 8	Baud Rate	9600
2B, 5F, 6, 6F	Baud Rate	19200
All Models	Data Bits	8
All Models	Stop Bits	1
All Models	Handshake	None
All Models	Parity	None

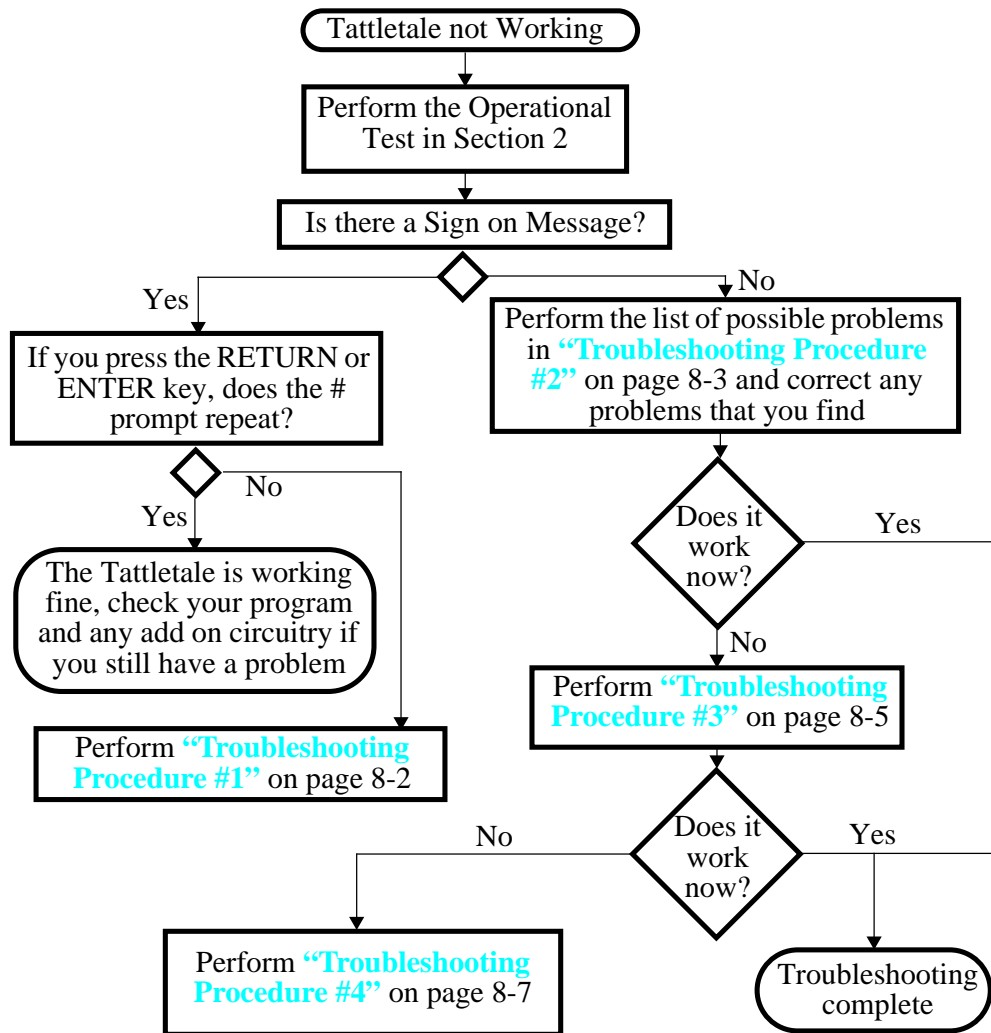


Figure 8-1: Troubleshooting Flow Chart

What to do if the Operation Test Fails

NOTE: Refer to Section 2 for the **Testing the Operation of the Tattletale** procedure.

Troubleshooting Procedure #1

If you get the sign-on message but cannot type any characters including the RETURN or ENTER key, perform the following:

1. Select QUIT from the File menu.
2. Disconnect the power source from the Tattletale.
3. Restart the TxTools application.
4. Connect the power source to the Tattletale. The screen should display the startup data again.
5. Press the RETURN or ENTER key and see if you get the # symbol to repeat.

6. If you still can't enter any characters, check the following:

NOTE: Try eliminating any circuits you added on to get back to the basic Tattletale, communication cable and computer.

- The battery or power supply may be weak under load. Check the battery with a 1K resistor across it.
- Check that the power supply current limit not set to <30mA.
- The power supply may be set to the wrong voltage. Set the voltage to 7 to 15 volts for the initial tests.
- The battery or power supply connection may be bad or intermittent. Wiggle connector and check VREG on the Tattletale.
- The communication cable may be bad. Check for a signal at the UART out line with an oscilloscope while powering up the Tattletale.

Troubleshooting Procedure #2

If you do NOT get the sign-on message any of the following may be the problem:

NOTE: Most of these can be tested by performing the “[Troubleshooting Procedure #3](#)” procedure on page 8-5.

- The wrong baud rate may be selected in the communications setup window (see Table 8-1).
- The default communication settings have been changed (see Table 8-1).
- The communication cable may be in the wrong port of the computer.
- The communication port may be set up for a mouse (IBM PC only).
- If a serial switch box is in use, disconnect it and connect the Tattletale communications cable directly to the serial port.
- The TxTools version may be different from the TxBASIC version burned into the EEPROM.
- The wrong port may be selected in the communications setup window.
- The notch on the Tattletale may not be aligned with the notch on the board (if the Tattletale and the prototyping board are equipped with the notch).
- The battery or power supply may be dead or weak under load.
- The power supply may be set to the wrong voltage.

- The battery or power supply connection may be bad or intermittent.
- The communication cable may be bad.
- The serial port on the computer may be bad. Check by shorting pins 2 & 3 of the Comm Port on the IBM PC or pins 3 & 5 on the Macintosh and typing characters on the keyboard.
- Try eliminating any circuits you added on to get back to the basic Tattletale, communication cable and computer.

NOTE: Make sure that you are using TxTools and that the Tattletale has TxBASiC burned into the EEPROM (both must be the same version). If you are using Tattletools and you have an OK> prompt instead of the # prompt, you are using TTBASiC, not TxBASiC.

If none of these suggestions work, try another computer or terminal if you can. If you still can't get the Tattletale to work, perform the [“Troubleshooting Procedure #4”](#) procedure on page 8-7.

NOTE: If you need to send anything back to Onset Computer, you will need to call Customer Support and get an RMA number.

After correcting the problem press the RETURN or ENTER key. The Tattletale should respond with another # symbol.

This simple test proves that the serial interface can send as well as receive. Once the # symbol repeats, you can proceed with:

[Section 3 - Operating the TxTools Program](#) for additional information on the TxTools software.

To write your own programs in TxBASiC refer to [Section 4 - Using TxBASiC](#) and [Section 5 - TxBASiC Command Reference](#). These sections show the details of all the commands available to operate the Tattletale.

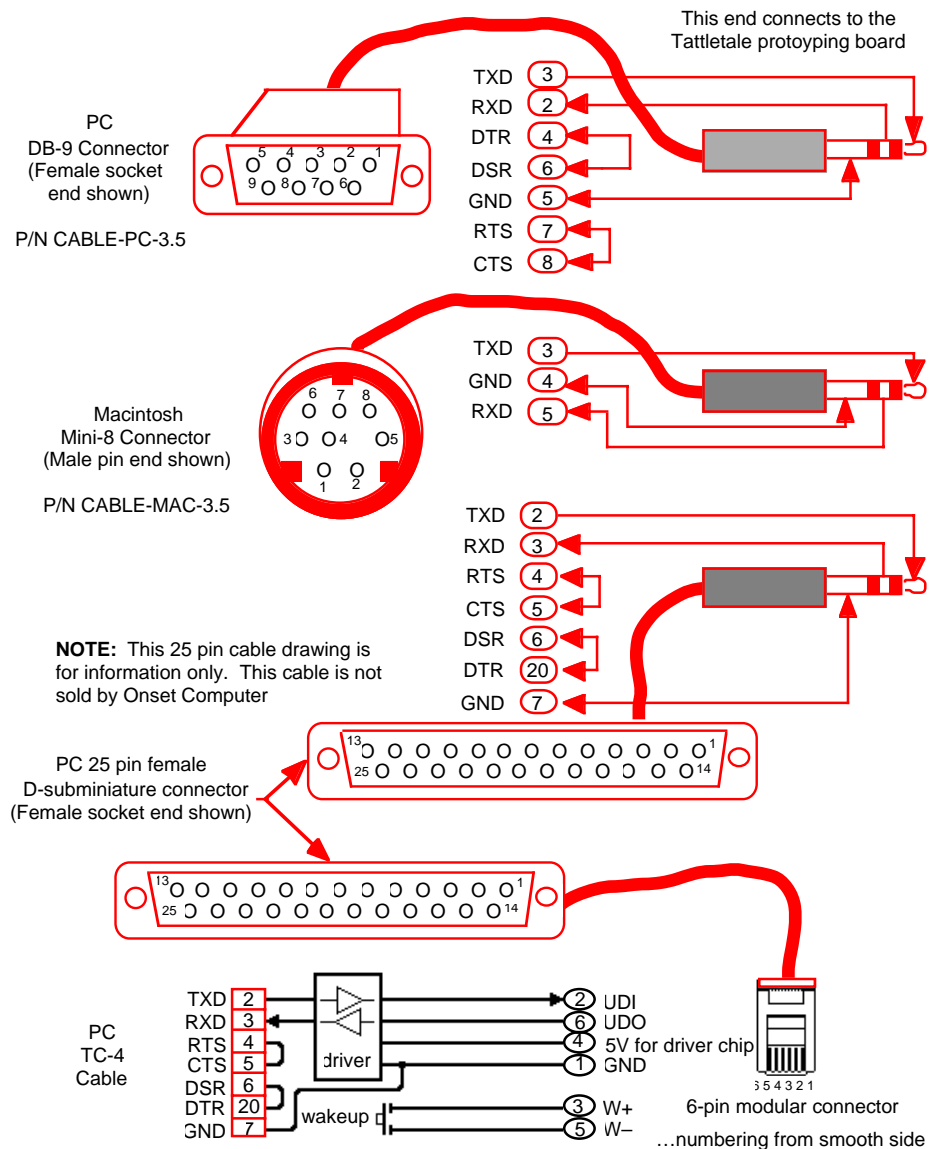


Figure 8-2: Communication Cable Pin Layouts

Troubleshooting Procedure #3

This procedure will verify the proper operation of the software, the computer's serial port and the communications cable without the Tattletale attached. It will help if someone else can hold the communications cable steady during this procedure.

NOTE: If you are using a TC-4 cable (with the modular connector) you will NOT be able to short the pins and run this test because the TC-4 cable needs power from the Tattletale for the driver chip in the cable itself.

NOTE: If this procedure does not work for either of the serial ports, then remove the communications cable from the back of the computer and short the TXD and RXD pins together (of the port selected in TxTools). Refer to Figure 8-2 for the locations of those pins. Since you are removing the cable from the computer, this will also work with configurations that use the TC-4 cable.

1. Verify that the communications cable is connected to one of the serial ports on the computer (IBM PC's: COM1 or COM2, Macintosh: Printer port or Phone port) and if the port is marked, record which one you are connected to.
2. Start the computer and start the TxTools software.
3. Select the CommPort menu (Terminal menu on Macintosh) and then select the Port Setup option (Baud Rate & Protocol option on Macintosh).

Verify that the port selected is the same port the communication cable is connected to. Also verify that the correct baud rate was selected (see Table 8-1) The values of the other parameters are not critical, but it is best if they are left at the default values. If you are not sure which port you are connected to then repeat the following steps for both of the serial ports.

NOTE: IBM PC's only - Be sure you're not trying to use the serial port used by a serial mouse. If you select the port your mouse is connected to you may lock up the computer.

- a. While using alligator clips, a piece of wire or even a bent paper clip, short the chrome tip of the phone plug to the next closest chrome section (see Figure 8-3).
- b. While shorting the plug, type characters on the keyboard. If everything is working correctly the characters typed should be displayed on the computer screen. If not, go back and select the other serial port and try again.
- c. If this procedure does not work for either of the serial ports, then the problem is either in the cable or the computer's serial port. If your computer has another serial port to connect to, repeat this procedure using it.
- d. If this procedure does display the characters typed on the screen and the Tattletale still will not communicate after reconnecting the Tattletale, perform **“Troubleshooting Procedure #4”** on page 8-7.

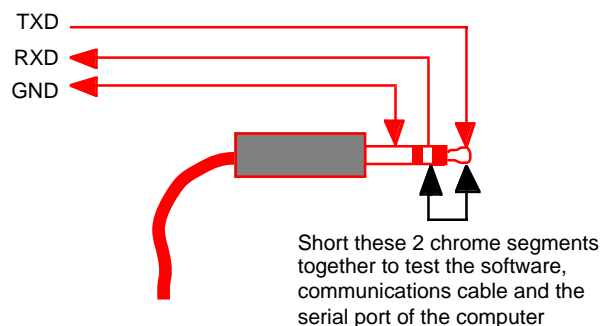


Figure 8-3: Testing the Communication Cable

Troubleshooting Procedure #4

NOTE: An oscilloscope and a DVM are required for this procedure. The Model 8 uses the label **VREF** in place of AD5V and **VREG** in place of the label REG5V.

The following is a very detailed procedure for measuring voltages and signals at specific pins on the Tattletale. This should only be performed if the previous 3 troubleshooting procedures have failed to solve your problem.

1. Check the baud rate. The baud rate should be set to the rate shown in Table 8-1 for your specific model.
2. Check the communication cable connections by disconnecting both ends of the communications cable and then insert the cable completely into the Tattletale and into the computer.
3. Make sure that the communication cable is connected to either ComPort 1 or ComPort 2 for IBM PC computers and either the printer port or the phone port on the Macintosh.

NOTE: IBM PC's only - Be sure you're not trying to use the serial port used by a serial mouse. If you select the port your mouse is connected to you may lock up the computer.

4. If your Tattletale has a notch in the board for locating pin 1, make sure that the notch on the Tattletale is aligned with the notch on the prototyping board. If the notches are not aligned, use the separation tool from the development kit (it looks like a large clamp) to remove the prototyping board from the Tattletale. It is possible that the Tattletale was damaged if power was applied while it was incorrectly mounted on the prototyping board.
5. Test the battery or power supply for the proper voltage (7 to 15V). If you are using a battery, test it under a load (with it connected to the Tattletale or put a 1K resistor across it) or you may think the battery is okay but in reality it is bad and supplying little or no current.

NOTE: The Tattletale power specifications state that the battery or power supply voltage can be from 6 to 15V which is correct; however, when you need to troubleshoot use a battery or power supply of 7 to 15V to help eliminate the power source as a potential problem.

6. With a DVM, measure between REG5V and DGND (see Table 8-2). There should be a 5V signal there.
7. With a DVM, measure between AD5V and DGND (see Table 8-2). There should be a 5V signal there also.

NOTE: Some Tattletaes only switch this on when taking an A-D measurement. The Model 8 turns VREF off while using the HYB command. (The sleep command leaves VREF turned on.) VREF is always on when the Model 8 is first turned on.

8. Using an oscilloscope, measure the ECLK signal (see Table 8-2). The signal should be a square wave from 0 to 5V and approximately 1.2MHz for slow crystals and 4.9MHz for fast crystals (the frequency is not important, just that it is actually putting out a signal is what we are looking for).
9. Using an oscilloscope, measure the UART Data Out at CMOS levels (see Table 8-2) and perform the following:
 - a. Turn off power to the Tattletale.
 - b. Turn on power to the Tattletale and look for any signal output at all. This is the pin that sends out the start-up message from the Tattletale, so any signal measured here, while turning the power on is a good sign that the Tattletale is trying to send out the sign-on data.
10. Using an oscilloscope, measure the UART Data Out at RS-232 levels (see Table 8-2) and perform the following:
 - a. Turn off power to the Tattletale.
 - b. Turn on power to the Tattletale and look for any signal output at all. Each time you power up the Tattletale, it will send the sign-on message out on this pin and you will see a digital signal on this pin switching between -4.5V and +4.5V. If you don't see a signal on this pin, the Tattletale may have a problem.

If you have performed all of these steps and still can't get your Tattletale to work, call Onset Computer customer service for an RMA # to send the Tattletale back for repairs.

Table 8-2: Troubleshooting Pins to Test

Model	REG5V Pin #	AD5V Pin #	ECLK Pin #	DGND Pin #	UART Data Out (CMOS Levels) Pin #	UART Data Out (RS-232 Levels) Pin #
Model 5	15	39	NC	20	24	TC-4, 25 pin end, pin 3
Model 5F	15	39	16	20	24	17
Model 5F-LCD	2	4	19	1	NC	On Connector
Model 6	15	31	17	16	25	TC-4, 25 pin end, pin 3
Model 6F	48	9	31	49	NC	14
Model 8	A2 (VREG)	B20 (VREF)	NC	A1	NC	A16, A14
NC = No connection available to test the signal at.						

Contacting Onset Computer Product Support

Please use the following methods to contact Onset Computer for support (they are in the preferred order):

1. Send a FAX
2. Send E-mail to Product Support over the Internet
3. Call Onset Computer Product Support

Regardless of which method you use to contact us, we will need the following information to help you:

- Tattletale model number
- Software version (TxTools and TxBASIC)
- Type of computer the Tattletale is connected to (IBM PC or Macintosh)
- Detailed information about the problem you are having
- Information about any added circuitry to the Tattletale or prototyping board

Sending a FAX to Product Support

If you have access to a FAX machine, you can easily contact us at (508) 759-9100. Please include as much detail about your problem and system configuration as possible.

Sending E-mail to Product Support over the Internet

If you have access to the internet you can send us E-mail at:

sales@onsetcomp.com
info@onsetcomp.com
support@onsetcomp.com

You can also visit our WORLD WIDE WEB page at: <http://www.onsetcomp.com>
or our anonymous FTP site at: <ftp://ftp.onsetcomp.com>

Calling Onset Computer Product Support

You can also call our Product Support Specialists at (508) 759-9500 between the hours of 8:30am - 5pm EST.

Appendix A - Manufacturer Contact Numbers

The following information is provided as an aid for helping you locate data sheets and other manufacturer specific information to design and build your product. This information was accurate at the time of printing, however; it is subject to change frequently.

Hitachi Corporation
Phone (800) 448-2244

Linear Technology
1630 McCarthy Blvd.
Milpitas, CA 95035-7417
Phone (408) 432-1900
Fax (408) 434-0507

Maxim Integrated Products, Inc.
120 San Gabriel Drive
Sunnyvale, CA 94086
Phone (800) 998-8800
Phone (408) 737-7600
Fax (408) 737-7194

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Phone (602) 786-7200
Fax (602) 899-9210

Motorola Literature Distribution
P.O. Box 20912
Phoenix, AZ 85036
Phone (800) 441-2447
Fax (303) 675-2150

Supertex Inc.
1350 Bordeaux Drive
Sunnyvale, CA 94089
Phone (408) 744-0100
Fax (408) 734-5247

Appendix B - Data Sheets

The following data sheets are provided to aid you in designing and building your products. Contact the specific manufacturer on the previous page for the latest information regarding any of these data sheets.

Glossary

Absolute reference	A voltage reference that does not vary with battery voltage.
A-D converter	Analog -to- digital converter - converts an analog signal to a digital code.
AGND	Analog ground reference - AGND should be tied directly to the analog ground plane.
ALTCLK	TxBASIC built-in subroutine to allow using Timer1 as the system clock, freeing Timer2 for use as a baud rate generator.
Amplifier	A circuit used to increase signal strength.
Analog inputs	The Tattletale inputs that accept data from analog sources.
Argument	A type of variable whose value is not a direct function of another variable. It can represent the location of a number in a mathematical operation, or the number with which a function works to produce its results.
Array	A set of related variables that are accessed by a common name with an index number.
Aux UART input	Auxiliary Universal Asynchronous Receiver Transmitter input
Aux UART output	Auxiliary Universal Asynchronous Receiver Transmitter output
Background task	A task that takes place simultaneously while a foreground task is being completed. Background task is rigorously timed. Foreground only executes when background is sleeping.
Baud rate	A unit of measurement of data communication speed. The speed in bauds is the number of signal elements per second. Since a signal element can represent more than one bit, baud is not synonymous with bits-per-second. Typical baud rates are 300, 1200, 2400, 4800, 9600 and 14,400.
Bipolar mode	An A-D mode where analog inputs can accept signals that may be positive or negative.
Block	An amount of storage space or data, arbitrary length, usually contiguous, and often composed of several similar records, all of which are handled as a unit.
Boot	Runs, usually at power up whatever program is burned into the ROM, EPROM or EEPROM.
Breakout Box	A device to put in a communications line that brings the signals out to test points without interrupting the signal.

COM	Common - The common pin defines the zero reference point for all single ended inputs. It must be free of noise and is usually tied to the analog ground plane.
Configuration byte	This byte provides important information to the Tattletale about default parameters on startup. It may be modified by the program.
Current drain	The amount of current a device uses to operate.
Data Bits	A logical 0, represented by 0V or a logical 1, represented by 5V. Eight bits together make up 1 byte.
Dual tasking	The ability to perform two tasks concurrently.
EEPROM	Electrically Erasable Programmable Read Only Memory - can be erased and reprogrammed with a power supply of the correct voltage and the host computer. Refer to EPROM and ROM for additional information.
EPROM	Erasable Programmable Read Only Memory - can be erased by a special EPROM ultraviolet eraser and then reprogrammed with the host computer. Refer to EEPROM and ROM for additional information.
FCB	Form Constant Byte - An assembler directive for initializing memory.
FCC	Form Constant Characters - An assembler directive for initializing memory.
FDB	Form Double Byte - An assembler directive for initializing memory.
Flash EEPROM	Erasable Programmable Read Only Memory - can be erased through software and then reprogrammed with the host computer. Refer to EEPROM and ROM for additional information.
Floating point arithmetic	A method of calculation in which the computer or program automatically records, and accounts for, the location of the radix point. The programmer need not consider the location of the radix point.
Foreground task	The task that spawns the background task. Background tasks are rigorously timed.
Gain error	The error in an amplifier that causes the output to be wrong by a constant multiplier.
Hall-effect	A magnetic field applied across a current-carrying material will force the moving carriers to crowd to one side of the conductor. An electric field will develop as a result of this crowding.
Handshake	The exchange of signals between communicating entities. Refer to protocol for additional information.
HOW?	When there is an error in a TxBASiC program it displays a TxTools error code number. The "HOW?" precedes the error number.
I/O lines	The digital input / output lines connected to the Tattletale.

Input impedance	The resistance an electronic device shows to the outside world. Impedance is really a complex combination of resistance, capacitance and inductance.
Input protection	Protects the Tattletale from voltages that would damage the Tattletale
Instrumentation amplifier	A very high gain amplifier with differential inputs. Has very low noise and high common mode rejection.
Interrupt	An asynchronous event that causes a computer program to interrupt what it is doing to service the event.
Labels	Since TxBASIC does not use line numbers, a text label is used at the beginning of command lines that require a reference, like a GOTO command.
LCD display	Liquid Crystal Display - The 4-character display used by the 5F-LCD.
Level shifting	To change the voltage of a signal for a specific purpose. It's almost like an amplifier. RS-232 level shifter we talk about converts 0V input to +5V and +5V input to -5V.
Linearity error	An error in an amplifier that causes the gain to vary over the range of inputs.
Logical operators	The Tattletale supports the two logical operators, AND (&) and OR (), which are used for both bit-wise operations and logical connectives.
Loss of precision error	A floating point error that results when a number with more significant digits than the floating point format can handle.
MAC-3.5	The cable used by the Macintosh to connect to the Tattletale.
Marking State	The marking state, also known as the idle state, is +5V (non-RS-232) and the negative voltage RS-232.
Memory map	A map showing the locations in memory for all the parameters, ROM and RAM data, and buffers etc.
Modem	A device that converts data from one form into another, as from one form usable in data processing to another form usable in telephonic transmission.
Monitor	The low level program that sends commands directly to the Tattletale (not accessible by the user).
NaN (Not-a-Number)	Not-a-Number is a floating point arithmetic result of an impossible operation - like taking the square root of a negative number.
Notches	Certain Tattletaes have a small notch in one corner for locating and matching pin 1 of the Tattletale with pin 1 of the prototyping board.
Operator	A symbol indicating an operation and itself the subject of the operation. It indicates the process that is being performed.

Output protection	A circuit used to protect the components connected to the output of the Tattletale.
Overflow error	The result is too large to be represented by the number system (either floating point or integer).
PC-3.5	The cable used to connect most IBM PC's to the Tattletale.
Parity	An extra bit used to detect errors in data sent over communication lines by making the sum of the active bits in a data word either an odd or an even number.
PCMCIA	Personal Computer Memory Card Industry Association - This is a memory card that stores data like a hard drive (but it has no moving parts) and can store a large amount of data on a card the size of a standard credit card.
Ping-pong buffering	Storing data to one part of a buffer while it's being read from another part.
Protocol	A formal set of conventions governing the format and relative timing of message exchange between two communicating processes. Refer to handshake for additional information.
Radix	A number that is made the fundamental number of a system of numbers; a base. Humans use a radix of 10, computers prefer 2.
RAM	Random Access Memory - A data storage device that can retain and produce on demand any data placed in it.
Ratiometric Measurement	To make a measurement as a percentage of its full-scale value (as opposed to an absolute measurement). Consistent readings are available even if the full-scale reference varies.
Real-time clock	A clock that keeps time in year, month, day, hour, minute, second format and updates it at some regular rate (usually one second).
Reference diode	Usually a Zener diode used to supply a voltage reference.
Reference input voltage (Voltage reference)	A fixed voltage used for comparing with other unknown or varying voltages.
Registers	Storage locations internal to the microprocessor that are used as operands for all actions.
Regulator, Voltage, REG5V	Keeps a steady amount of voltage over a range of current drains being output from the voltage regulator so long as the input voltage to the voltage regulator does not drop below some cut-off voltage.
Remind	The command used to copy off of the Tattletale and store the program and operating system for later burning into EEPROM

ReMinder	The device sold and recommended by Onset Computer Corporation for programming the older Tattletales that are not equipped with EEPROMS.
Resistivity	How resistant something is to allow electricity to flow.
ROM	Read Only Memory - A data storage device that permanently retains all data placed in it and produces that data on demand. Once the data is placed into it, it can't be changed. Refer to EEPROM and EPROM for additional information.
Schottky diode	A very fast switching diode with a low turn-on threshold (0.4V or less versus the normal 0.6V of other diodes).
SCR latchup	If the input (or output) is driven beyond the supply momentarily.
Serial transmission	A system in which the data bits occur serially in time.
Setup time	The amount of time for a system, device or program to reach a required initial condition.
Soldering pencil	A soldering iron with a very fine point for soldering in tight areas.
Square wave	A signal with only two stable states that alternates between these states so that it spends equal time in each state.
Start Bit	This is the first bit issued when a UART wants to send a data byte. The start bit will unambiguously tell the receiver that a set of data bits is to follow. This is normally a space, or 0 bit, which is 0V (non-RS-232) and a positive voltage for RS-232.
Stop Bits	This bit is normally issued by a UART to follow the data byte. This bit is usually a mark, or a 1 bit, and is used to separate consecutive characters and to help with resynchronization.
String operators	Symbols representing the various operations that can be performed on strings - comparing, copying, appending etc.
Symbol table	A list of all the variables, arrays and labels used in a program. Usually, addresses and sizes of the symbols are included.
Tabs	A group of ASCII tab characters. Also called the horizontal tab or HT. The ASCII character value is 9.
Thermistor	An electronic component used to measure temperature. It is also much easier to interface to than a thermocouple.
Thermocouple	A bi-metal device used to measure temperatures of a much greater range than a thermistor.
Tokenizer flags	A group of flags that determine actions taken by the tokenizer as well as the type and format of the files of files generated during a syntax check.
TxBASIC	The high level software language used to program the Tattletale.

TxTools	This software provides a window environment for creating programs in TxBASIC.
UART	Universal Asynchronous Receiver Transmitter - A UART is normally a single chip dedicated to converting bytes to and from a serial data stream that can travel on RS-232 data lines.
Underflow error	Floating point math error resulting from trying to represent a number closer to zero than the format allows.
VPT-1	An add-on board for Tattletale Lite or Model 5F-LCD.
Waiting for NAK	A state of the XMODEM file transfer protocol where it is waiting for the character (ASCII NAK = 15H) signaling to start.
XMODEM	A simple file transfer protocol used to send and receive files.

Index

Section Number – Page Number

Symbols

! (exclamation point)	4-15
(backslash) ...	5-19, 5-67, 5-73, 5-88, 5-97, 5-106
# (pound sign)	4-13, 4-14
#02 (format specifier)	4-24
% (percent)	4-1, 4-14
& (ampersand)	4-15
&H (hexadecimal prefix)	4-3, 4-22
' (apostrophe)	4-22
(colon)	4-21, 4-28, 4-35
(semicolon)	4-18
(spaces)	4-14
* (asterisk)	4-12, 4-26, 5-84, 5-91
. (period)	4-15
// (double slash)	4-2, 4-19
? array	4-23, 4-46, 5-83, 5-95
? variable	4-44, 4-46, 5-95
@ array	4-55, 5-31
(digital logic operator)	4-15
~ (tilde)	4-26, 4-27, 5-84, 5-91

Numerics

10-bit A-D	5-18, 5-22
12-bit A-D	5-18, 5-22

A

Abbreviations	4-1
ABS	4-47, 5-4, 5-9
A-D converter	1-3, 6-17
ADLOOP	4-1, 4-3, 4-45, 4-57, 5-2, 5-10
ADLOOP array	4-57, 5-10
ADoff	5-1, 5-3
ADon	5-1, 5-2
AGND	6-17
AIN	4-47, 5-4, 5-13
ALIGN	4-40
Alt-B	3-4, 3-15
Alt-C	3-4, 3-18
ALTCLK	4-42
Alt-D	3-4, 3-12
Alt-E	3-4, 3-11
Alt-F	3-4, 3-9
Alt-L	3-4, 3-14, 4-64

Alt-M	3-4, 3-16
Alt-O	3-4, 3-15
Alt-P	3-4, 3-19
Alt-Q	3-4, 3-9
Alt-R	3-4, 3-14
Alt-S	3-4, 3-13
Alt-T	3-4, 3-14
Alt-V	3-4, 3-16
Analog inputs	5-22, 5-23, 6-17, 7-3, 7-5
AND	4-1, 4-4, G-3
ANIN	4-41, 4-42
Arithmetic	4-48
Array	4-45
ASFLT	5-4, 5-14
ASM	5-2, 5-15
Assembler opcodes	4-29
Assembly Language	4-28
Assembly language subroutines	4-41
ATN	5-4, 5-17
Aux UART input	6-16
Aux UART output	6-16

B

Background task	4-25, 5-10, 5-84, 5-96
BAT	4-53
Battery	2-1, 4-53, 4-62, 6-3, 7-3, 8-3
Baud rate	3-20, 3-39, 3-42, 3-44, 4-42, 5-105, 5-106, 5-1, 5-2, 5-10, 5-11, 5-25, 6-5, 6-7, 6-18, 7-8, 7-19, 7-22, 8-1, G-1
Bipolar	5-1, 5-3
Bipolar mode	7-10
Blank line	4-4, 4-19, 4-21
Boot	4-8, 4-59, 5-21, 5-66
Break	4-2, 4-59
Build Your Own Basic (BYOB)	4-1
BURST	4-3, 5-4, 5-12, 5-18
Burst2KSetup	5-1, 5-3
BurstAD	5-1, 5-4
BurstInfo	5-1, 5-5

C

CALL	5-2, 5-19
CBREAK	5-2, 5-21
CHAN	5-4, 5-22
Character constants	5-8
Command reference (part 1)	5-2
Command reference (part 2)	5-1

Configuration byte 4-42, G-2
 Connectors 6-1
 Constants 4-47
 Conversions 4-49
 Converting TTBASIC programs 4-51
 ConvertVarPtr 5-1, 5-5
 COS 5-4, 5-25
 COUNT 5-2, 5-4, 5-26
 CtrlCHandle 5-1, 5-5
 CtrlCReset 5-1, 5-6
 Current drain 6-15, 6-19

D

Datafile 4-17
 DFMAX 4-45
 DFREAD 5-28
 DFSAVE 5-4, 5-29
 DFSIZE 5-2, 5-30
 DIM 5-2, 5-31
 DISPLY 5-32
 DOFFLD 5-3, 5-33
 DONE 5-34
 DSKOFF 5-3, 5-35
 DSKON 5-3, 5-35
 DTOA 5-3, 5-36
 Dual tasking 4-25

E

Editing 3-5, 4-3
 EEPROM 2-4, 4-7, 4-61, 4-62, 4-64, 4-65
 ELSE 5-3
 ENDIF 5-3, 5-49
 EOFF 4-53
 EON 4-53
 EPROM 4-59
 Erasing problems 4-65
 Erasing the Tattletale 4-62
 Error handling 4-3, 4-5, 5-3, 5-65
 Error messages 4-24
 Errors, tokenizer 4-24
 EXP 4-47, 5-4, 5-37

F

FCB 4-38
 FCC 4-38
 FDB 4-38
 FIX 4-25, 4-47, 5-4, 5-38

Float 5-4, 5-13, 5-14, 5-38, 5-39, 5-52
 Floating point 5-39
 Floating point errors 4-3, 4-50
 FLUSH 4-41
 FOR 4-19, 5-3, 5-40
 Foreground task 5-96
 Formats 5-62, 5-73
 FPERR 4-45, 4-50
 FRE 4-53
 Functions 4-47

G

GET 4-22, 4-24, 5-4, 5-42
 GETMEM 4-41
 GETS 5-4, 5-7, 5-43
 GetTickRate 5-1, 5-6
 GOSUB 4-4, 4-10, 4-54, 5-3, 5-44
 GOTO 4-12, 4-54, 5-3, 5-45

H

HALT 5-6, 5-46
 Hexadecimal 4-3, 4-22
 HOW 4-24
 HPSleep 5-1, 5-6
 HYB 5-6, 5-47
 HybAt3V 5-1, 5-7
 HybCheckForBREAK 5-1, 5-7

I

I/O lines 6-13
 I/O-8 board 6-1
 IF 4-18, 5-3, 5-48
 IFF 5-3, 5-49
 INPUT 4-9, 4-19, 4-25, 4-58, 5-3, 5-50
 Input protection 7-15, 7-16
 Input, analog 7-17
 Input, digital 7-17
 INSTR 5-4, 5-7, 5-51
 Instrumentation amplifier 7-11, 7-14
 INT 5-4, 5-52
 Interrupt 4-43, 4-58
 ITEXT 4-3, 4-22, 4-25, 5-4, 5-53

K

KbChar 5-1, 5-8
 KbFlush 5-1, 5-8
 KbHit 5-1, 5-8

L

Labels4-2, 4-4, 4-28, 4-52, 5-15
 LABPTR4-48, 5-4, 5-55
 LCD display5-88, 7-1
 LEFT5-4, 5-7, 5-56
 LEN5-5, 5-7, 5-57
 LET5-3, 5-58
 LIST4-24, 4-53
 Load4-8
 Loading programs4-59
 LOG5-5, 5-59
 LOG104-48, 5-5, 5-60
 Logical operators4-4
 Loss of precision error4-50
 LPMODE5-1, 5-9
 LT1077, 1078, 10797-13

M

Memory map4-44
 MID5-5, 5-7, 5-61
 MODEL4-45, 5-3, 5-62
 Monitor4-7
 Multiple statements4-4

N

NaN4-46
 NEW4-53
 NEXT4-10, 4-19, 4-58
 Not-a-Number error5-59, 5-60
 Notches8-7

O

OFFLD5-4, 5-63
 Offloading programs4-61
 ONERR4-3, 4-21, 5-3, 5-65
 ONLOAD4-53
 Operators4-1, 4-4, 4-5
 OR4-1, 4-4, G-3
 OTEXT4-22, 5-4, 5-67
 Output protection7-15
 Overflow4-4, 4-24
 Overflow error4-4

P

PC-3.56-5
 PCLR5-5, 5-68
 PEEK5-5, 5-69

PEEKL5-5, 5-69
 PEEKW5-5, 5-69
 PERIOD5-5, 5-70
 PIN4-19, 4-23, 4-54, 5-5, 5-71
 Ping-pong buffering5-29
 POKE5-3, 5-72
 POKEL5-3, 5-72
 POKEW5-3, 5-72
 Power-up4-59
 Precedence4-1, 4-48
 PRINT4-12, 5-3, 5-73
 Print/Store formats4-50
 PSET5-5, 5-75
 PTOG5-5, 5-76
 Pulse width4-32

Q

QUIT5-3, 5-77
 Quotes4-39, 5-50, 5-73

R

Radix4-35, 5-16
 RAM6-20
 RATE4-25, 5-3, 5-78
 Ratiometric A-D5-22
 Read-only variables4-45
 Real-time clock4-43, 5-6, 5-83, 5-95
 Registers4-42, 5-2, 5-86
 Relational operations4-48
 REM4-53, 5-3, 5-80
 Remind4-60, 4-61, 4-64
 ReMinder4-60
 REPEAT5-3, 5-81
 RES4-38, 4-40
 Reset5-1, 5-10
 RETURN4-10, 5-3
 RIGHT5-5, 5-7, 5-82
 RMB4-38, 4-40
 RTI4-29, 5-11
 RTIME5-6, 5-83
 RUN5-84
 RUN label5-3

S

Schottky diode7-10
 SCROLL5-85
 SDI5-5, 5-86

- SDO 5-5, 5-88
- SerActivate 5-1, 5-10
- SerGetBaud 5-1, 5-10
- SerSetBaud 5-1, 5-11
- SerShutDown 5-1, 5-11
- Setup time 5-23
- SimGetFSys 5-1, 5-12
- SimSetFSys 5-1, 5-12
- SIN 5-5, 5-90
- SLEEP 4-12, 5-6, 5-91
- SQR 5-5, 5-94
- Square wave 4-23, 5-103
- STEP 5-3, 5-40
- STIME 4-24, 5-6, 5-95
- STOP 4-21, 5-3, 5-96
- StopWatchStart 5-1, 5-14
- StopWatchTime 5-2, 5-15
- STORE 4-18, 4-24, 4-50, 5-4, 5-97
- STR 5-5, 5-7, 5-99
- String functions 5-6
- String handling operations 5-7
- Strings 5-73
- Strings in TxBASIC 5-6
- STRMEM 4-41
- SWI 4-43, 4-58
- Symbol table 3-18, 3-36, 3-37, 3-38
- T**
- Tabs 3-31, 4-5
- TAN 5-5, 5-100
- TEMP 4-12, 5-5, 5-101
- Thermistor 1-9, 2-1
- Time 5-6
- Timing 4-5, 5-88, 5-91
- TLC1078 7-13
- Token list 4-24, 5-65
- Tokenizer flags 3-17
- TONE 4-23, 4-25, 5-5, 5-103
- TPUAccumPeriod 5-2, 5-15
- TPUAccumPulseWidth 5-2, 5-16
- TPUChangePWM 5-2, 5-17
- TPUCountChan 5-2, 5-17
- TPUGetTCR1 5-2, 5-17
- TPUoff 5-2, 5-18
- TPUon 5-2, 5-18
- TPUPeriodChan 5-2, 5-18
- TPUSdiClkChan 5-2, 5-18
- TPUSdiDatChan 5-2, 5-19
- TPUSdiLchChan 5-2, 5-19
- TPUSdoClkChan 5-2, 5-19
- TPUSdoDatChan 5-2, 5-20
- TPUSdoLchChan 5-2, 5-20
- TPUSetupPWM 5-2, 5-21
- TPUToneChan 5-2, 5-20
- TPUUgetChan 5-2, 5-21
- TPUUsendChan 5-2, 5-21
- Trace 4-42, 5-2, 5-15
- TSerByteAvail 5-2, 5-22
- TSerClose 5-2, 5-22
- TSerGetByte 5-2, 5-23
- TSerInFlush 5-2, 5-23
- TSerOpen 5-2, 5-23
- TSerPutByte 5-2, 5-24
- TSerResetBaud 5-2, 5-25
- TSTOUT 4-42
- TxBASIC 4-1
- TxBASIC floating point 4-46
- TxBASIC variables 4-44
- TxTools 3-1
- U**
- UART 6-5, 6-16, 6-18
- UGET 4-24, 5-4, 5-105
- Underflow error 5-37
- UNTIL 5-3, 5-81
- UPROG 4-57
- URTGET 4-41
- URTSND 4-41
- URTTST 4-41
- USEND 4-25, 5-4, 5-106
- V**
- VAL 5-5, 5-7, 5-108
- VARPTR 5-5, 5-109
- VERS 4-45
- VGET 4-24, 4-55, 5-5, 5-110
- VSTORE 4-24, 4-55, 5-3, 5-111
- W**
- Waiting for NAK 5-63
- WEND 5-3, 5-112
- WHILE 5-3, 5-112
- White space 4-6, 4-37

X

XMIT-5-3, 5-113
XMIT+5-3, 5-113
XMODEM 4-8
XSHAKE5-3, 5-114

Tattletale Model 8-TXB

Installation and Operation Manual

Revision History

<u>Revision</u>	<u>Release Date</u>	<u>Description of revision or changes</u>
D-EAS-00008 Rev. A	October 1995	Initial release of the new Model 8 manual. This is the first Tattletale Model 8 manual that was written specifically for TxBASIC users.
D-EAS-00008 Rev. B	December 1995	Updated the entire index, replaced pages 2-7, 5-3, 5-4, 5-6, 5-32, 5-36, 5-51, 5-56, 5-61, 5-82, 5-85 and 6-24. Also replaced the LTC-1174 data sheets.
D-EAS-00008 Rev. C	July 1997	Pages 1-2, 6-1 and 6-7 were updated by adding new information on the Operating Temperature range and updating the specifications in Table 1-3 and Table 6-1. Page 6-1 had a correction to the pin layout for the SquishyBus.
D-3284-A	July 1998	Updated for modifications in TxBASIC version 4.10.
D-3284-B	February 1999	Updated line: Interconnects are available in various heights to accommodate your external hardware, an example of an elaxtomic interconnect is the Fujipoly's part number 0940020 which is 0.250in high.
D-3284-C	September 1999	Page 5-83 and 5-95 remove reference to commands not working on 6F and Model 8. Update schematic Fig. 6-6, Page 6-6 to Rev C. Page 6-24, add note PCMCIA Card Adapter obsolete.

