JagBASIC®
for JAGXTREME® Terminals

Programmer's Guide

16384600A
(10/01)

CUSTOMER FEEDBACK

Your feedback is important to us! If you have a problem with this product or its documentation, or a suggestion on how we can serve you better, please fill out and send this form to us. Or, send your feedback via email to: quality_feedback.mtwt@mt.com. If you are in the United States, you can mail this postpaid form to the address on the reverse side or fax it to (614) 438-4355. If you are outside the United States, please apply the appropriate amount of postage before mailing.

Your Name:	Date:
Organization Name:	Mettler Toledo Order Number
Address:	Part / Product Name:
	Part / Model Number:
	Serial Number:
Phone Number: ( ) Fax Number: ( )	Company Name of Installation:
E-mail Address:	Contact Name:
	Phone Number:

How well did this product meet your expectations in its intended use?		Comments:
	Met and exceeded my needs	
	Met all needs	
	Met most needs	
	Met some needs	
	Did not meet my needs	

PROBLEM:

UNACCEPTABLE DELIVERY:

- ☐ Shipped late
- ☐ Shipped early
- ☐ Shipped to incorrect location
- ☐ Other (Please Specify)

OUT OF BOX ERROR:

- ☐ Wrong item
- ☐ Wrong part
- ☐ Missing equipment
- ☐ Equipment failure

- ☐ Wrong documentation
- ☐ Missing documentation
- ☐ Incorrectly calibrated
- ☐ Other (Please specify)

Comments:

DO NOT WRITE IN SPACE BELOW; FOR METTLER TOLEDO USE ONLY

☐ Retail ☐ Light Industrial ☐ Heavy Industrial ☐ Systems

RESPONSE: Include Root Cause Analysis and Corrective Action Taken.

FOLD THIS FLAP FIRST

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 414 COLUMBUS, OH

POSTAGE WILL BE PAID BY ADDRESSEE

Mettler-Toledo, Inc.
Quality Manager - MTWI
P.O. Box 1705
Columbus, OH 43216
USA

Please seal with tape.

INTRODUCTION

This publication is provided solely as a guide for individuals who have received Technical Training in servicing the METTLER TOLEDO product.

Information regarding METTLER TOLEDO Technical Training may be obtained by writing to:

METTLER TOLEDO
1900 Polaris Parkway
Columbus, Ohio 43240
(US and Canada) 614- 438-4511
(All Others) 614-438-4888

FCC Notice

This device complies with Part 15 of the FCC Rules and the Radio Interference Requirements of the Canadian Department of Communications. Operation is subject to the following conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his or her own expense.

METTLER TOLEDO RESERVES THE RIGHT TO MAKE REFINEMENTS OR CHANGES WITHOUT NOTICE.

PRECAUTIONS

READ this manual BEFORE operating or servicing this equipment.

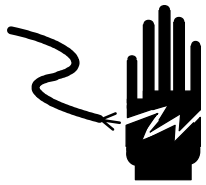
FOLLOW these instructions carefully.

SAVE this manual for future reference.

DO NOT allow untrained personnel to operate, clean, inspect, maintain, service, or tamper with this equipment.

ALWAYS DISCONNECT this equipment from the power source before cleaning or performing maintenance.

CALL METTLER TOLEDO for parts, information, and service.

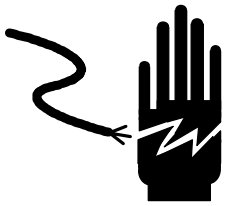


⚠ WARNING

DISCONNECT ALL POWER TO THIS UNIT BEFORE INSTALLING, SERVICING, CLEANING, OR REMOVING THE FUSE. FAILURE TO DO SO COULD RESULT IN BODILY HARM AND/OR PROPERTY DAMAGE.

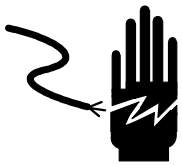
⚠ CAUTION

OBSERVE PRECAUTIONS FOR HANDLING ELECTROSTATIC SENSITIVE DEVICES.



⚠ WARNING

PERMIT ONLY QUALIFIED PERSONNEL TO SERVICE THIS EQUIPMENT. EXERCISE CARE WHEN MAKING CHECKS, TESTS AND ADJUSTMENTS THAT MUST BE MADE WITH POWER ON. FAILING TO OBSERVE THESE PRECAUTIONS CAN RESULT IN BODILY HARM OR EQUIPMENT DAMAGE.



⚠ WARNING

FOR CONTINUED PROTECTION AGAINST SHOCK HAZARD, CONNECT TO PROPERLY GROUNDED OUTLET ONLY. DO NOT REMOVE THE GROUND PRONG.

⚠ CAUTION

BEFORE CONNECTING OR DISCONNECTING ANY INTERNAL ELECTRONIC COMPONENTS OR INTERCONNECTING WIRING BETWEEN ELECTRONIC EQUIPMENT, ALWAYS REMOVE POWER AND WAIT AT LEAST THIRTY (30) SECONDS BEFORE ANY CONNECTIONS OR DISCONNECTION'S ARE MADE. FAILURE TO OBSERVE THESE PRECAUTIONS COULD RESULT IN DAMAGE TO OR DESTRUCTION OF THE EQUIPMENT, OR BODILY HARM.

CONTENTS

1 Introduction

Overview

NOTE: The information in this manual is specifically for use with JAGXTREME terminals. For information on using JagBASIC with JAGUAR terminals, please refer to the JagBASIC manuals with part numbers C14839600A or earlier (non-revision, A revision, or B revision).

JagBASIC is a tool for customizing the JAGXTREME industrial scale terminal. It provides the means for creating custom operator interaction for data input using the JAGXTREME terminal's 16-character lower display and keypad. An external keyboard, serially connected display devices, as well as the terminal display, may be used to communicate messages to the operator.

Programming Language

The JagBASIC language is a standard BASIC programming language with more than 120 standard BASIC statements and functions, plus extensions for special JAGXTREME terminal operations. The language provides functionality to perform many tasks including operator interaction, serial input and output, discrete input and output, scale data exchange, string manipulation, arithmetical and relational operations, and open, close, read (get) and write (put) file operations.

Editors

JagBASIC includes a simple line editor that uses the JAGXTREME terminal's lower display. When the "BasTerminal" is selected in the serial port setup menu, a remote PC with a terminal emulator program interfaces with the line editor. These editors permit creation and modification of JagBASIC programs.

Entering, Editing and Managing Programs

When using JagBASIC file names of file1.bas through file9.bas, the operator may start any of nine programs by pressing the FUNCTION key followed by the program number. This provides a simple way to manage multiple programs as separate modes of operation and allows larger applications to be divided into smaller, more manageable programs. The file1.bas program may be designated to automatically start on power-up. Other file names can be used but must be called up using the JagBASIC LOAD command, or chained from the main JagBASIC program.

Small JagBASIC programs may be entered and edited on the terminal with an external keyboard using the lower display. This allows simple programs to be quickly entered or modifications to larger programs to be made in the field without additional equipment. A personal computer (PC) is recommended when creating larger programs. A PC can be directly connected to the terminal through a serial port. The PC, running a terminal emulator, acts as a monitor and keyboard for the terminal. Using Zmodem protocol over a serial port or FTP on a JAGXTREME terminal through Ethernet, files can be transferred between the PC and the terminal.

JagBASIC Integration and Security in the JAGXTREME Terminal

JagBASIC is integrated into the operating environment of the JAGXTREME terminal. JagBASIC programs reside with the standard terminal program. The JagBASIC interpreter runs as a separate task using the terminal's multi-tasking operating system. This allows the custom JagBASIC program to interact with other terminal tasks and resources using the terminal's exclusive shared memory design. All shared memory in the JAGXTREME terminal may be accessed by the JagBASIC program using this simple construction.

JagBASIC programs are stored as source files then interpreted in the terminal. Source file storage allows you to edit the program on the terminal and provides the security of having the source available even if a PC stored copy is not available. The source files may be retrieved from the terminal for archiving, modification, or duplication. The JagBASIC interpreter was designed to provide a more secure operating environment where the program is restricted from accessing and possibly corrupting the standard functions of the terminal. Access to JagBASIC can be password-protected to limit access to the source code, or the operator may be given access to all of the standard terminal functions as well as the custom functions provided by the JagBASIC program.

JagBASIC Encryption Feature

The JagBASIC program encryption feature prevents unauthorized users from modifying or copying a JagBASIC program. The program developer can encrypt the JagBASIC program using the either JagBASIC preprocessor or the new PC JagBASIC programming utility. The encrypted program file has the ".cpt" name qualifier. When the program is loaded, the JagBASIC interpreter automatically decrypts the encrypted program before running it. You cannot save or list the encrypted program using the JagBASIC interpreter. Also, you cannot extract an encrypted program from the JAGXTREME terminal using ZMODEM or FTP. Using the JagBasic Preprocessor, place a "-e" on the command line to encrypt the program. For example,

jbpp filetest.bas file1.cpt -N1 -I1 –e

Using PCJagBASIC Editor, run the preprocessor with the option for output code with encryption selected.

Compatibility with Scales and PLCs

JagBASIC will operate in JAGXTREME terminals configured for any type of scale, including METTLER TOLEDO's DigiTOL® bench/portable scales, high precision scales, floor scales, truck scales, or industry standard analog load cell scales such as tank or hopper weighing systems. JagBASIC can also co-exist and communicate through shared memory with PLC interfaces.

File Transfer

JagBASIC programs and data files are stored in the terminal in a DOS file-compatible RAMDISK. An FTP communications utility permits these files to be sent between a JAGXTREME terminal and a PC, using FTP utility software sent over Ethernet. The optional PCJagBASIC Editor's built-in file transfer utility enables files to be sent serially between a JAGXTREME terminal and a PC.

File Specifications

The JAGXTREME terminal file system has a capacity of 1900 KB for program and data files. The maximum number of files is 96. Individual programs have a limit of 600 lines of code or 30 KB. The maximum line length and string size is 160 characters. The maximum number of variables is 200.

If the JAGXTREME terminal's alibi memory is enabled, the RAMDISK space available to JagBASIC is reduced to 900K bytes. Before configuring alibi memory, it is recommended that all files be backed up since you will likely lose files in the RAMDISK.

Standards Compliance

JagBASIC is based on the American National Standards Institute (ANSI) standard for minimal BASIC (ANSI X3.60-1978) with extensions and integration into the JAGXTREME terminal operating environment by METTLER TOLEDO. Programmers familiar with BASIC can quickly become proficient in using JagBASIC.

PC Program Development

METTLER TOLEDO's PCJagBASIC Editor (P/N 09170301) is recommended for program development. PCJagBASIC is a tool for programming, debugging, and archiving JagBASIC programs. PCJagBASIC Editor features:

- Multiple code windows
- Code development without line numbers – label and procedure name support Document access
- Data access
- Preprocessor with setup selection for JAGXTREME or custom versions
- Alias filename support
- Built-in file upload/download
- JagBASIC command, shared data, and macro help
- Total project archival
- Reference code windows
- New macros: table, I/O, timer, If/Then/EndIf, While, Close
- Built-in debug window

NOTES

2 Shared Data

JAGXTREME Terminal Operating Environment and Shared Data

Refer to Chapter 6 of this manual for information on Shared Data Variables.

Three concepts are fundamental to the way the JAGXTREME terminal handles data within the terminal's operating system: Shared Data Database, Shared Data Callbacks, and an Event Driven Ladder Logic Engine. They enable the terminal to:

- Handle a multitude of actions virtually concurrently,
- Provide fast reaction to internal and external instructions, and
- Provide users with maximum flexibility to meet their application demands.

Shared Data Database

Central to the JAGXTREME terminal's open architecture is the implementation of a Shared Data Database. This central table of variables tracks virtually every data value used within the terminal. Variables containing values corresponding to weight information, setup and calibration parameters, user input literals, prompts and responses, printer templates and setpoint information are all stored in this table. The status of physical and logical discrete inputs and outputs as well as the "mappings" of serial and discrete I/O connections are also stored. The terminal accesses and uses this database as a central depository for information used in all functions related to:

- Weighing and process control
- Communication with external printers, bar code readers, and other devices
- Personal computer hosts
- PLCs
- Applications written in the JagBASIC programming language

Shared Data Callbacks

The JAGXTREME terminal couples this database concept with Shared Data Callbacks. Each operating system task has the shared data variables it uses mapped directly to it. Whenever a task requires a specific variable or group of variables, their values can be found in the Shared Data Table. Every time a shared data variable is changed, all operating system tasks, which use the variable, are identified and notified that a change has occurred using the Shared Data Callback. When a particular task is notified of a change, the task is executed, updating all other affected variables and related tasks.

For example, if a logical 1 is written to the discrete logical variable associated with the pushbutton tare command, the scale task is notified and the process of initiating a tare undertaken. This affects the shared data variables associated with the displayed weight, tare weight, and net weight, among others. Changes in these variables initiate other tasks and affect other variables (e.g. the variable associated with whether or not a net weight is being displayed). This automatic processing of tasks simplifies interfacing the terminal to external controls such as a PLC, or to an internal JagBASIC program.

Event Driven Ladder Logic Engine

In traditional ladder logic circuits, an engine continually cycles through the rungs of the ladder, allowing any changes in the coils to cause a change in the contacts. Wherever an input is changed, the corresponding output is potentially changed to reflect the change in input. The JAGXTREME terminal uses the Event Driven Ladder Logic Engine concept to scan for changes in shared data variables ("coils" or inputs) and to make resulting changes in other shared data variables, the terminal outputs or displays (contacts or outputs).

The terminal's ladder logic creates a smart ladder logic engine. The terminal's engine only runs when an event occurs. The event or triggering mechanism could be a change in a shared data variable, a JAGXTREME terminal message, or the result of some type of physical input. Once the ladder logic engine is run, the changes cause an "output engine" to run and make changes in shared data variables, physical outputs, and/or terminal messages. These may, in turn, cause further "cycling" of the ladder logic engine and result in further changes.

Shared Data Types

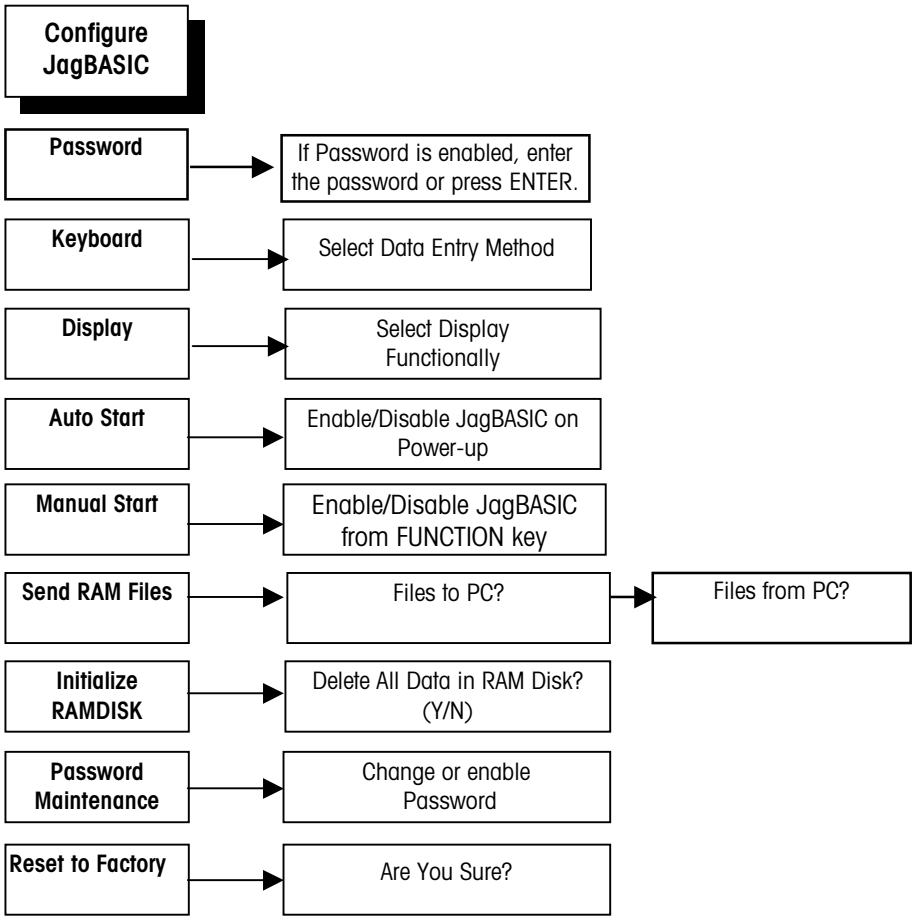
There are four types of Shared Data Variables.

- The first holds the values associated with different scale parameters such as displayed weight and tare weight. These variables function like fields in a database. The fields stored include setpoint values, time and date information, and user programmed literals and prompts. The actual values stored in these variables may be strings, integers, or double precision floating point numbers. Besides these values, status or source information may be stored.
- The second type of variable is a level-sensitive logical variable. These values store a logical 1 or 0 as an integer in a bit field within shared data. These particular variables are known as "level-sensitive" because they generate a callback when either a 0 or a 1 is written to the field. These variables indicate the status of a particular scale condition, such as whether a particular scale is in motion or over capacity, or whether or not a particular setpoint is feeding or a weight is within a setpoint tolerance. By reading the values of these variables, the programmer can determine the status of a particular trait of the terminal without having to use an actual terminal discrete output.
- The third variable type is an edge-sensitive logical variable. A logical 1 or 0 is stored as an integer in a shared data bit field. These variables differ from those above in that they trigger a callback when a 1 is written to the field. When the "triggered" task is complete, a 0 is automatically written (by the terminal) back to the field. In terms of some of the operations of the terminal, a 1 written to one of these variables would be like pressing a button on the terminal front panel. By using these variables, the programmer could initiate a scale task in the same way as if a pushbutton was pressed or a discrete input were used.
- The last variable type indicates the status of the physical discrete inputs and outputs found on the Controller and multi-function boards. The stored logical 1s or 0s correspond to whether a physical discrete input or output is true or false, on or off. It may be useful to use these variables to initiate further actions within a program in conjunction with an external event tied to a physical input or output.

3 Setup

Configuring JagBASIC in the JAGXTREME Terminal

The JAGXTREME terminal setup contains a special program block and sub-blocks for configuring JagBASIC, as shown here.



SW2-2 must be OFF for normal JagBASIC operation.

Note: Use an anti-static strap when touching the controller PCB.

To access the program block, you must first enter setup by pressing the FUNCTION key and then SELECT until **[Enter Setup?]** is displayed. Press ENTER.

Press the SELECT key until **[Config JagBASIC]** is displayed. Press ENTER.

At the **[Passwd?]** prompt, enter a password. Password security allows the JagBASIC programs to be protected from unauthorized changes. Press ENTER**.** Or, just press ENTER if no password has been previously configured. Entry of an incorrect password will cause the terminal to display the message **[Access Denied.]**

At the **[Keyboard]** prompt (which permits designation of the keyboard input device that will pass characters to a JagBASIC program when an INPUT or INKEY statement is executed and of the device that will be used for the BASIC command line mode) press ENTER to access the sub-block and then press SELECT to choose:

- **[None]**—No keyboard input is required. This would be used with programs that monitor other I/O then act in the background without operator intervention.
- **[Keypad]**—The terminal keypad is used only for operator input to JagBASIC. The normal JAGXTREME keypad functions are not available.
- **[Kboard]**—External QWERTY keyboard or remote PC with terminal emulator attached to the terminal will be used for operator input to JagBASIC.
- **[Both]**—Both the terminal keypad and an external keyboard will input to JagBASIC.

At the **[Display]** prompt (which permits designation of the display output device that will be used by a JagBASIC program when a PRINT statement is executed), press ENTER to access the sub-block and then press SELECT to choose:

- **[None]**—No display output device is to be used.
- **[JAGUAR]*—**The terminal's lower display is to be used. This display will also be used for standard terminal functions.

At the **[Autostart?]** prompt (which enables or disables the automatic start up of the file1.bas JagBASIC program on power up) press ENTER to access the sub-block and then press SELECT to choose **Y(es)** to use the automatic program start feature or **N(o)** to disable it. If you select **Y(es),** JagBASIC will automatically start file1.bas **program** on power up and when you exit setup.

If you selected **N(o)** the **[Manual Start]** prompt appears, which allows you enable or disable the manual mode startup of JagBASIC programs by pressing the FUNCTION key. Select keys 1 to 9 to represent file1.bas through file9.bas.

At the **[Send RAM Files]** prompt, press ENTER to access the sub-block.

At the **[Files to PC? N]** prompt, press ENTER to select **N(o)** or press SELECT and then ENTER to select **Y(es).**

- If you choose **Y(es)**, the terminal prompts you with **[Are You Sure?]**. Choose **Y(es)** to place the terminal in the mode to transmit its RAMDISK files to a PC.
- If you choose **N(o)**, you will be prompted with **[Files From PC?].**
  - If you select **N(o),** you will go the next prompt.
  - If you select **Y(es),** you will see the prompt **[Are You Sure?]**
    - If you select **Y(es),** you will place the terminal in a mode to receive files from a PC. The terminal will display **[Recving from PC.]** The file transfer is initiated from the PC. Refer to the chapter on programming fundamentals for details of this operation. If communication with the PC is not established, the terminal will time out and return to the sub-block.

At the **[Init RAM Disk?]** prompt, you can delete all files in the terminal's RAMDISK. Press ENTER to access the sub-block. The terminal will then prompt with **[Are You Sure?]** You must then choose **Y(es)** to delete the RAMDISK files. The display will read **[Please Standby]** and then the system will reboot before displaying **[BASIC]**. You must re-enter setup and scroll through the sub-blocks until you reach the **[Password Maint]** prompt.

Make sure the password is written down in a secure place. If the password is lost, the only way to re-enter the JagBASIC Configuration menu is by performing a Master Reset which will erase all configuration data in the terminal and set all values to factory defaults!

You will also lose JagBASIC files stored on the ramdisk when a Master Reset is performed. Do not perform a Master Reset unless you can reload the JagBASIC files!

*At the **[Display]** prompt, the choices are None or JAGUAR, with JAGUAR representing the JAGXTREME terminal.

NOTE: Use caution when selecting this option since the files cannot be recovered once they are deleted!

Once you enter a password, be sure to record it in a secure place and provide it to all persons who will need to access the JagBASIC program block.

At this prompt, you can configure a security password to be configured for the JagBASIC programs. Press ENTER to access the sub-block. The terminal will then prompt with **[Passwd?]** Enter a password of up to eight characters and then press ENTER. After exiting the program block this time, you will need this password to re-enter the block.

Reset to Factory?—This sub-block allows you to reset the Config JagBASIC program block parameters to their factory settings.

Connecting the Terminal to a PC

Refer to the following diagram for proper cable connections to the terminal's serial ports COM1 and COM2. COM1 and COM2 are located on the Controller board, which is positioned in the top slot.

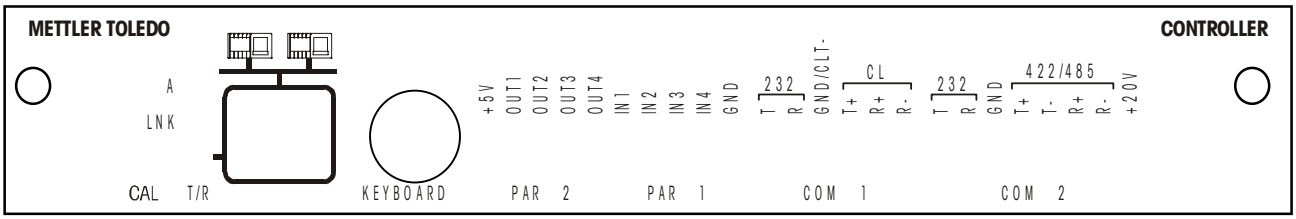


Figure 3-1: Controller PCB Rear View

The COM1 and COM2 terminal strips will accommodate wire sizes ranging from 23 to 16 AWG. The terminal strips may be removed to facilitate wiring. Removal of the terminal strips also permits easier viewing of the terminal designations printed on the board back plate.

The following diagram and table describe COM1 (or COM2) pin-to-pin cable connections using an RS-232 cable to a PC serial port. The maximum recommended cable length for RS-232 communications is 50 feet (15.24 meters.)

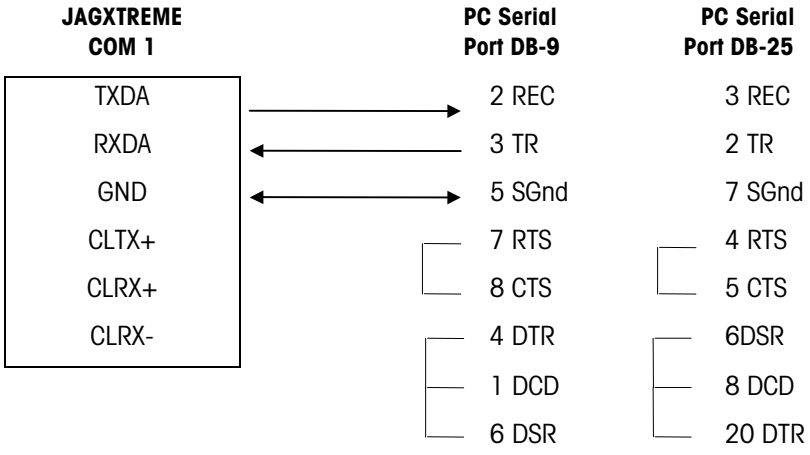


Figure 3-2: RS232 Connections to Terminal and PC Serial Port

The PC cable can be used for five different applications:

- Flashing new software into the terminal through COM2.
- JagBASIC file transfer through COM1.
- LPRINT device output to a terminal emulation program or communications program to receive data sent using LPRINT, LIST, VARS, etc. Output from the terminal will be sent to the first port configured for demand output.
- PCJagBASIC Editor allows a PC terminal emulator to act as a program development interface for JagBASIC.
- JagBASIC program interface directly to serial ports for input and output.

You can connect the JAGXTREME terminal to a personal computer (PC) using either of these methods:

- A 10 BASE-T Category 5 Cross-over cable.
- An Ethernet Hub and standard patch cables.

4 Programming Fundamentals

JagBASIC Files

JagBASIC program files are stored in the JAGXTREME terminal's battery backed RAMDISK file system. This file system is equivalent to the file system on a PC.

Naming Conventions

JagBASIC enables you to run nine program files using the function key followed by a digit. The files are named as follows:

- file1.bas
- file2.bas
- file3.bas
- file4.bas
- file5.bas
- file6.bas
- file7.bas
- file8.bas
- file9.bas

Throughout the documentation, these files are referred to as filex.bas files. When using other names, the names must follow the MS-DOS file name conventions -- an 8-character prefix and 3-character suffix (normally .bas). Characters A-Z and 0-9 can be used. Some characters are reserved and cannot be used in file names, such as #, ^, %, *, (,), {, }, [,]. These files can only be called up and run using the BASIC interpreter commands RUN or LOAD, or called from within another program using the CHAIN command. See Chapter 5, JagBASIC Commands, for more information.

Program Size

The maximum size for a program is 600 lines and 30 KB. The maximum number of variables is 200. The maximum string size and line size is 160 bytes.

Line Numbers

JagBASIC requires the use of line numbers for every line of the BASIC program. Programs that start line numbering at 1 and are numbered sequentially (1, 2, 3, 4, etc.) execute most efficiently on the JAGXTREME terminal. The largest line number permitted is 30,000. The JagBASIC preprocessor supports symbolic labels and automatically numbers the program lines.

Line Length

Line lengths are limited to 160 characters per line, therefore string sizes are limited 160 characters also.

Multiple Statements on a Line

Unless noted, you can put multiple statements on a line if they are separated by a colon (:). The program will be more legible if only single statements are placed on a line.

Data Files

You can use data file numbers from 0 to 7 with JagBASIC. The JAGXTREME terminal's RAMDISK has 1900 KB of file space available. Using the JAGXTREME terminal's alibi memory feature reduces the amount of RAM disk space to 900 KB and a maximum of 96 RAM disk files.

Operator and Program Controls

Automatic Startup

Selecting Automatic Startup in the JagBASIC configuration allows a hands-off, power-on startup of the JAGXTREME terminal. When Automatic Startup is selected, the file1.bas program runs at startup and on exit from setup mode. The JAGXTREME terminal does not auto-start the JagBASIC program when existing setup. JagBASIC must be manually restarted by an operation.

Starting JagBASIC Programs

The nine JagBASIC programs file1.bas through file9.bas may be started by pressing the FUNCTION key followed by the program number. For example, to run file3.bas, press the FUNCTION key then the number **3**. This provides a simple way of managing multiple programs as separate modes of operation and allows larger applications to be divided into more manageable programs.

Stopping JagBASIC Programs

Every program written in JagBASIC should include an END statement to cause termination. A program may be stopped at any time by pressing the ESC key twice, as long as the EXC key has not been disabled by the JagBASIC code. A program will automatically terminate any time the JAGXTREME terminal is placed into setup from the terminal's front panel.

Note: A JagBASIC program will not automatically terminate if remote access is attempted through the embedded web server. The JagBASIC program will have to be stopped before remote access to setup is permitted.

Interaction of JagBASIC Program with JAGXTREME Web Browser

The JAGXTREME web browser will not permit a user to enter setup when there is a JagBASIC program running. The JAGXTREME shared data variable t_61e is set to "1" to indicate to the JagBASIC program that the web browser operator is attempting to enter setup. The JagBASIC program may optionally monitor this variable and terminate itself. Once the program terminates, the web browser operator can then instruct the JAGXTREME to enter setup. An operator at the JAGXTREME can also manually start JagBASIC through the function keys or by cycling power at the JAGXTREME terminal.

Switching the Display between JagBASIC and the Terminal

To disable the stop program (press ESC twice) and switch display (press SELECT) functionalities, write to the shared data variables /bas86, /bas89, and /bas87, respectively, in the program file.

While a JagBASIC program is running, press the ESC key once to assign the lower JagBASIC display back to the JAGXTREME terminal. JagBASIC will continue to run. Only the display is changed. To return to the JagBASIC display, press SELECT.

Securing a JagBASIC Program

Unencrypted JagBASIC programs can be secured so that a user cannot alter or illegally procure a program. To secure a program:

1. Set the password in the JagBASIC Setup menus.
2. Set AutoStart=Y in the Setup menus.
3. Within the JagBASIC program, set Manual Stop Enable(bas89)=0. This prevents a user from stopping the program.
4. Name your startup program file1.bas.

Refer to page 1-2 for information on the JagBASIC program encryption option.

Using the Terminal BASIC Interpreter

You may use a PC terminal emulator or the terminal's display and an external keyboard to create and edit JagBASIC programs. Programs are entered at the JagBASIC interpreter prompt. With JagBASIC enabled and no programs running, press the **ESC** key to display the interpreter "BASIC:" prompt. From this prompt you may start typing lines of BASIC or type in a BASIC command. Entering a line of code to the interpreter without a line number will cause the interpreter to execute the line immediately.

Creating and Editing JagBASIC Program Files

Using PCJagBASIC

PCJagBASIC is a self-contained development environment that handles the editing, debugging, and file management of JagBASIC programs. The software is Windows-compliant and contains application help files.

Using a PC

You may use a personal computer (PC) to create and edit the JagBASIC program file using either a DOS or Windows text editor. Files must use standard DOS attributes, such as date, time, length, and reserved characters.

When you have completed writing the program in the text editor, send the text file to the terminal's COM1 serial port using one of the following:

- A communications program such as RIPterm©
- HyperTerminal© if using Windows 95 or higher.
- Procomm Plus for Windows

The text file will be stored in the RAMDISK. The file transfer uses standard Zmodem file transfer protocol. The JagBASIC RZ command initiates receiving files at the terminal from the PC using the ZMODEM protocol over the BasTerminal serial communication line. The JagBASIC SZ command initiates sending files from the terminal to the PC. If you want to use the RZ and SZ commands from the BasTerminal, you need to set up the serial communications to use the 8-bit, No Parity data format

File Transfers

For file transfers, setup the PC for 8-bit, No Parity, 1 Stop Bit. These settings are independent of the serial port settings in the terminal. The file names will be displayed on the terminal's lower display as they are transferred. Always start the file transfer process on the PC, then on the terminal.

Sending Files to the Terminal

Set the serial port to 9600,8,N,1 to match the file transfer fixed settings of 9600,8,N,1. This enables you to upload and download files, plus receive output into your communications program without requiring any parameter changes.

The terminal will always use 9600,8,N,1 for file transfer, overriding the serial port defaults.

The JAGXTREME terminal is capable of receiving files using standard Zmodem file transfer protocol.

To send files to the terminal from your PC:

1. Set the password in the JagBASIC Setup menus (optional).
2. Select Zmodem protocol at the PC communications utility.
3. Type in or select the file, but do not start the transfer.
4. Set up the terminal for receiving files.
    - If you have a PC console for JagBASIC, type "**RZ**" at the JagBASIC interpreter prompt and then press ENTER. Proceed to step 8. You do not need to complete step 9.
    - If you do not have a PC console for JagBASIC, press the FUNCITON key. Press SELECT until [**Enter Setup?]** is displayed and then press ENTER. Proceed to step 5.

5. Press SELECT until **[Config JagBASIC]** is displayed, then press ENTER.
6. When **[Passwd?]** is displayed, enter the password. If no password has been programmed, press ENTER.
7. Press SELECT until **[Send RAM Files]** displays, then press ENTER. Press ENTER again when **[Files To PC? N]** displays. When **[Files From PC? N]** is displayed, press SELECT to change the prompt to **Y(es)**, and then press ENTER again.
8. Start the communications program file transfer.
   - If using RIPterm, press the Page Up key, select **Zmodem** then type in the file name.
   - If using HyperTerminal, click **Transfer, Send File**, then type in the file name, or use browse to locate the file. When the file has been selected, click **OK**.
9. When the PC file transfer has been started, press ENTER on the terminal keyboard to start the transfer. As the files are sent to the terminal, the file names will display on the lower terminal display.

Receiving Files from the Terminal

Set the serial port to 9600, 8N1 to match the file transfer fixed settings of 9600,8,N,1. This enables you to upload and download files, plus receive output into your communications program without requiring any parameter changes. The terminal will always use 9600,8,N,1 for file transfer, overriding the serial port defaults.

The PC receives all unencrypted files currently residing in the terminal RAMDISK, including BASIC files and any data files that exist. Any encrypted file (extension of .cpt) cannot be sent to the PC.

To set up the terminal to send files to the PC:

1. If you have a PC console for JagBASIC, type "SZ" at the JagBASIC interpreter prompt and then press ENTER. Proceed to step 5; however, you do not need to press ENTER in step 5.

   Or, if you do not have a PC console for JagBASIC, press the FUNCTION key, then SELECT until **[Enter Setup?]** displays. Then press ENTER.
2. Press SELECT until **[Config JagBASIC]** is displayed, then press ENTER.
3. When **[Passwd?]** is displayed, enter the password, or if no password has been programmed, just press ENTER.
4. Press SELECT until **[Send RAM Files]** is displayed, then press ENTER. When **[Files To PC? N]** is displayed, press SELECT to change the prompt to **Y**, then press ENTER again to display **[Are You Sure? N]**. Press SELECT to change the **N** to **Y**.
5. If the autodownload function is not enabled in your communications software, start the download in the PC software program, then press ENTER on the terminal keyboard to start the transfer.

Using the JagBASIC Preprocessor

The JagBASIC preprocessor can translate from the free line format permitted in PC BASIC to the strict line numbering format required by JagBASIC, strip out memory consuming comments (REM statements), and warn of JagBASIC constraint violations. The JagBASIC preprocessor is available as part of the JagBASIC programmers kit.

User Environment

The JagBASIC preprocessor is a DOS-based, command line oriented utility. It is invoked with command line arguments as follows: jbpp infile outfile (option)

infile Input text file, with free format statement labels.

outfile Output text file, with JagBASIC line numbering, and error messages.

option Any of a combination of command line options, including the following:

-R Pass through all REM statements from the infile to the outfile. Default is eliminating REM statements from the output file.

-NXXX Start statement numbering with XXX. Default starting number is 100.

-IYYY Increment statement numbering by YYY. Default increment step is 10.

-W Do not compress white space within a statement. Default is to compress multiple consecutive space (or tab) characters to a single space (or tab) character.

The output file is suitable to be downloaded to a JagBASIC enabled terminal.

Example: jbpp bulkway.bas filel.bas -N1 -I1.

Run Time Operation

The primary purpose of the preprocessor is to add line numbers to all statements and to replace symbolic labels with numeric labels. Two passes through the input file are required. The first creates a list of symbolic labels. The second adds line numbers and performs error checking on the resulting output file. Symbolic labels are typically identified as the first single word on a line that is followed by a colon. In the following example, the "begin" is a symbolic label.

```
Example 1: IF x = 1 THEN GOTO BEGIN
           y = 1
           begin:
```

```
Example 2: GOSUB CheckMotion
           .
           .
           CheckMotion:
           RETURN
```

The JagBASIC preprocessor identifies symbols that are preceded by an "xx" as symbolic labels, allowing the JagBASIC program to build state tables within the program. The symbols in the following statement are interpreted as symbolic labels.

```
Example:  FillCycle:
          .
          .
          CloseGates:
          .
          .
          CaptureGross:
          .
          .
          RecordGross:
          .
          .
          DATA xx FillCycle, xx CloseGates, xx CatureGross, xx RecordGross
```

Line Number Substitution

Label numbering normally uses 100 as the first statement in the JagBASIC program and increments statement numbers by 10. Both defaults can be overridden using optional command line arguments. One or more blank lines encountered in the source file causes the next line number to be adjusted upward to the next nearest module 100. Line numbers encountered in the input file are treated just like other symbolic labels and are substituted accordingly.

Programs execute most efficiently if 1 is set as the first line number and subsequent line numbers are incremented by 1.

White Space, Blank Line, and Comment Handling

Multiple consecutive space (or tab) characters encountered within an input file statement are compressed to a single space (or tab) character unless the user specified otherwise via optional command line argument. One or more consecutive blank lines encountered in the input file are output as a single blank line in the output file. Remarks (REM statements) are eliminated unless instructed otherwise by the user via an optional command line switch.

Error Checking

Several JagBASIC specific error conditions are checked in the preprocessor. In each case, an error message is added to the output file on a new line following the line containing the error. The error message is also output to the console. A count of total errors is provided on the console and at the end of the output file at the completion of the preprocessor. No error count message is added to the output file if no errors are detected.

Exceeding the maximum number of lines or maximum program size are fatal errors. Preprocessor operation stops at the first occurrence of a fatal error condition.

General Error Messages

The preprocessor can return the following general error messages:

"Error** Label Not Found! Input File Line #"**

—When a GOTO or GOSUB is followed by a label, the label should appear in the JagBASIC file.

"Error** Maximum Char. Per Line(80) Reached! Input File Line #"**

—The maximum characters per line are 80 characters.

"Error** Duplicate Label Found! Input File Line #"**

—A label was previously found in the document. The second label is ignored.

Fatal Error Messages

The preprocessor can return the following fatal errors. The preprocessor terminates when the first fatal error is encountered.

"Error** No Label! Input File Line #"**

—When a GOTO or GOSUB is present a label must follow the GOTO or GOSUB.

"Error** Maximum Line #(30000) Reached! Input File Line #"**

—The maximum line number is 30,000.

"Error** Maximum Number Of Output Lines Reached(600)! Input File Line #"**

—The maximum number of lines allowed in the output file is 600.

"Error** Maximum Output File Size Reached(30000 Bytes)!"**

—The maximum byte size of the output file is 30,000 bytes.

Serial Terminal Support

JagBASIC supports a serial terminal, such as a dumb terminal or a PC running a terminal emulator, as a console for JagBASIC program development and debugging. You can type commands at the keyboard and view the typed commands on the serial terminal display. The serial terminal must be attached to a serial port on the local terminal. BasTerminal must be assigned to the serial port in the Serial Config menus. BasTerminal is also used for the debug window in PCJagBASIC Editor.

Configuring BasTerminal

The Configure Serial menus allow you to setup the JagBASIC keyboard input from a serial port. Select the appropriate port and assign the BasTerminal connection. Input characters from the serial port are routed to JagBASIC. This connection is for keyboard input to the JagBASIC interpreter. The BASIC interpreter displays the "BASIC:" prompt and input keystrokes to the BasTerminal. You must assign the keyboard to JagBASIC in the JagBASIC setup menus. To transfer files from the PC to JagBASIC, use 8 bits, no parity.

TPRINT Command

You can output messages to the BasTerminal from a BASIC application using the TPRINT command. It has the same syntax as the PRINT and LPRINT commands. This is a simple program for entering data and echoing it to BasTerminal using the INKEY$ function and TPRINT.

```
10 PRINT "enter line"
30 c$=INKEY$
40 IF C$="" THEN GOTO 30
50 IF C$=CHR$(08) THEN GOTO 90
60 TPRINT c$;
70 x$=x$+c$
80 GOTO 30
90 TPRINT ""
100 TPRINT "input line= ";x$
110 GOTO 10
```

Configuring LPRINT Device

The LPRINT device is the first demand print port assigned to Scale A in the serial setup menus. In a typical development setup, both BasTerminal and LPRINT device would be assigned to Com Port 1. Com Port 1 is also the default Zmodem file transfer port.

Special Keys

BasTerminal translates the following standard serial input keys to these terminal internal key values. You can use the following keys on a standard serial keyboard to simulate the function keys on JAGXTREME keypad.

Serial Input Character			JAGXTREME Character (Hex Value)	
Back Space	(0x08)	is translated to	Delete	(0x7f)
Tab	(0x09)	is translated to	Select	(0x05)
Escape	(0x1b)	is translated to	Escape	(0x02)
Enter	(0x0d)	is translated to	Enter	(0x08)
Ctrl+A	(0x01)	is translated to	Function	(0x01)
Ctrl+B	(0x02)	is translated to	Escape	(0x02)
Ctrl+C	(0x03)	is translated to	Memory	(0x03)
Ctrl+D	(0x04)	is translated to	Tare	(0x04)
Ctrl+E	(0x05)	is translated to	Select	(0x05)
Ctrl+F	(0x06)	is translated to	Clear	(0x06)
Ctrl+G	(0x07)	is translated to	Zero	(0x07)
Ctrl+H	(0x08)	is translated to	Enter	(0x08)

NOTES

5

JagBASIC Commands

The JagBASIC commands are broken into 12 groups:

Interpreter Commands—perform file and program maintenance functions, transfer files, and aid in debugging.

Variable Commands—assign values to variables, define global variables, exchange variable values, access the terminal's shared database, declare arrays, read values from a DATA statement and assign them to variables, and allow DATA statements to be reread from a specified line.

Flow Control and Operator Commands—repeat a section of the program; branch to a specified line number; execute a sub-statement depending on specified conditions; provide logical operators for use in decision statements; clear the JagBASIC execution stacks; send program control to the first line of the current program, and branch to a location specified by a variable's value.

Math Commands—execute trigonometric, logarithmic and exponential, conversion, rounding and truncation, random number generating, and other arithmetic operations.

String Commands—extract part of a string, convert decimal numbers to hexadecimal or octal numbers, convert a character to ASCII code and the reverse, create "filler" strings, count the number of characters in a string or the number of bytes required to store a variable, display the string representation of a number, locate one string within another string, and interpret the string entered by the user as though it were a number.

Simple I/O Commands—sound the terminal beeper on a specified input or output, generate prompts, accept user input from the keyboard, check for key presses, and format output with tabs and spaces.

Serial I/O Commands—access files; open or close a serial port; flush received data in the BIOS serial input buffer; read input from the keyboard or serial port; output data to a terminal serial COMx port; print formatted output on the LPRINT device; output data to the specified serial port; and assign an output line width to the LPRINT device or a file.

File Commands—open and close a file; convert strings to numbers and the reverse; read, write, and delete records from the indexed sequential file; test for the end of a file; allocate space for variables in a random-access file buffer; identify files as indexed sequential files; identify which field in a record is the index key; read and write to a sequential file; get records from and put records in an indexed file; read all characters of an entire line; return the current position within a file; and move data into a random-access file buffer.

Real-Time Process Control Commands—allocate and de-allocate events; suspend program execution until an event trigger causes program execution to resume; clear outstanding event triggers; disable asynchronous event triggers; re-enable asynchronous event triggers after a critical section of code; return the state of an event; enable asynchronously monitoring of an event; enable ladder logic rungs.

Timing Commands—set or return the terminal system date and time; suspend program execution for the of specified number of milliseconds; start and stop the timer; and return a double precision floating point number that contains the elapsed time in seconds.

Error Trapping Commands—return the runtime error code for the most recent error; return the line number where the error occurred, or the closest line number before the line where the error occurred; simulate an occurrence of an error; and enable error handling and, when an error occurs, directs your program to an error handling routine.

TCP/IP Commands – allow JagBASIC application programs to use TCP/IP communications. (Only available in JAGXTREME terminals.)

Each group contains examples and information on the command's usage and syntax. Some commands are discussed in two places in the chapter since they apply to more than one area. JagBASIC syntax and program examples use the following conventions:

Commands are not case-sensitive.

Square brackets [ ] signify optional information.

Divider bars | signify the available choices.

Interpreter Commands

JagBASIC programs can be entered in the JagBASIC interpreter. The interpreter provides a secure operating environment where programs are restricted from accessing and corrupting the terminal's standard functions. Interpreter commands are typed at the "BASIC:" prompt to perform a function. With JagBASIC enabled and no programs running, press **ESC** to get the BASIC: prompt. Start typing lines of JagBASIC or a JagBASIC command. The interpreter's program and file maintenance commands enable you to:

- Close all files, release file buffers, clear all common variables, set numeric variables and arrays to zero, and set string variables to null.
- End a program and close all files.
- Delete a specific program line or a range of lines.
- Display the RAMDISK directory on the LPRINT device.
- Free the memory used by an array.
- Load and delete files from the RAMDISK.
- Save the current BASIC program to the RAMDISK with the specified file name.
- List all variables, or all or part of the program, to the LPRINT device.
- Clear the current program and all variables from memory.
- Execute the current file in memory.
- Terminate program execution and return to command level.
- Add comments or reference remarks to the code listing.
- Trace program execution for debugging purposes.
- Initiate a Zmodem file receive or transfer between serial port 1 and the RAMDISK.
- Insert/clear breakpoints
- Insert/clear watchpoints
- Resume execution after break
- Step execution after break
- Show variable's name and current value

This section discusses the following JagBASIC interpreter commands:

Command	Usage
BREAK	Stops execution of program at line number
CLEAR	Closes files, releases file buffers, clears common variables, sets numeric variables and arrays to 0, sets string variables to null. Can also be used to clear BREAK or WATCH.
DELETE	Deletes a specific program line or a range of lines.
DIR	Prints the RAMDISK directory on the LPRINT device.
END	Ends a program and closes all files.
ERASE	Frees the memory used by an array.
KILL	Deletes the specified file from the terminal RAMDISK.
LIST	Lists all or part of a program to the LPRINT device.
LOAD	Loads a file (filename.bas) from the RAMDISK into memory.
NEXTLINE	Displays next line number to execute or sets a "new" next line
NEW	Clears the current program and all variables from memory.
REM	Enables you to add any comments or reference remarks to the code listing.
RESET KEYS	Recovers control of the keypad and keyboard for program editing or accessing setup.
RUN	Executes the current file in memory or resumes execution after a BREAK.
RZ	Initiates a Zmodem file receive over serial port 1 into the RAMDISK file system.
SAVE	Saves the current BASIC program in memory to the RAMDISK with the specified file name.
SHOW	Displays last line executed, the variable name and current value
STEP	Executes next line number after BREAK
STOP	Terminates program execution; returns to command level or executes temporary break.
SZ	Initiates a Zmodem file transfer over serial port 1 from the RAMDISK.
TRON TROFF	Enables and disables tracing of program statements.
VARS	Prints a list of all variables to the LPRINT device.
WATCH	Monitors variable during execution

BREAK

Usage
A breakpoint is 1 of up to 20 line numbers. JagBASIC compares the breakpoints with the next line number to execute. If JagBASIC finds a match, it stops execution and displays the next line number to execute in square brackets. You can remove a breakpoint from the list with the BREAK OFF option or by clearing all breakpoints with the CLEAR or CLEAR BREAK statements.

The BREAK command without any parameters displays the current list of all breakpoints.

Syntax
BREAK linenumber [OFF]

Example
BREAK 100

CLEAR

Usage
Closes all files, releases file buffers, clears all common variables, sets numeric variables and arrays to zero, and sets string variables to null. Used to reinitialize all variables to zero or to null. Clears all break and watch points.

Syntax
CLEAR [BREAK I WATCH]

Example
CLEAR (Clears both break points and watch points)

CLEAR BREAK (Clears break points only)

CLEAR WATCH (Clears watch points only)

DELETE

Usage
Deletes a specific program line or a range of lines.

Syntax
DELETE line[-line]

line The number of the line in the program that you want to delete. If a range of lines is deleted, the first, the last, and all lines inclusive in the range are deleted.

Example 1
DELETE 40

Example 2
DELETE 40-100

DIR

Usage
Prints the RAMDISK directory on the LPRINT device.

Syntax
DIR

Example
DIR

END

Usage
Ends a program and closes all files. If a program contains subroutines, an END statement should be placed between the main program and the first subroutine to prevent you from inadvertently running the subroutine. An END statement is executed implicitly at the end of every program.

Syntax
END

Example 1
10 PRINT "Program Over."
20 END

Example 2
520 IF K>1000 THEN END ELSE GOTO 20

ERASE

Usage
Frees the memory used by an array. Arrays may be redimensioned after they are erased so the memory space allocated may be used for other purposes.

Syntax
ERASE array name [,array name]...

array name The name of the array that you want to erase from memory.

Example
200 DIM B(250)

.

.

.

450 ERASE B

KILL

Usage
Deletes the specified file from the terminal RAMDISK and frees the space it occupied.

Syntax
KILL "filename.bas"

filename.bas The name of the file that you want to delete.

Example 1
KILL "file4.bas"

Example 2
10 KILL "data2.bas"

LIST

Usage
Lists all or part of a program to the LPRINT device.

Syntax
LIST [startline-endline]

Startline Range of line numbers that you want to list to the LPRINT device.

endline Startline is the first line to print and endline is the last line to print. If startline and endline are not specified, the entire program will be listed.

Example 1
LIST

Example 2
LIST 10-20

LOAD

Usage
Loads a file from the RAMDISK into memory. LOAD closes all open files and deletes all variables residing in memory before loading the new file.

Syntax
LOAD "filename.bas"

filename.bas The name of the file that you want to load into memory. If the extension and end quotes are omitted, .bas is assumed.

Example 1
LOAD "file1.bas"

Example 2
LOAD "TEST

NEW

Usage
Clears the current program and all variables from memory.

Syntax
NEW

Example
NEW

NEXTLINE

Usage
NEXTLINE displays the next line number to execute or send a new next line when you set the optional line number. JagBASIC displays the new line number to confirm the selection. If you enter a nonexistent line number, JagBASIC retains the current next line. Square brackets surround the displayed line number. You use RUN to resume execution. If you use NEXTLINE to reposition the program into or out of an execution block, you will likely get a program execution error after execution resumes. If there is no program currently executing, NEXTLINE displays a [0] line number.

Syntax
NEXTLINE [linenumber]

Example
NEXTLINE

NEXTLINE 100

RESETKEYS

Usage
The RESETKEYS command sets the JagBASIC keyboard parameters back to a known state. (Autostart to "N" and Manual Start to ""N".) It is used primarily when debugging a program that takes control of the keyboard parameters. If the program crashes, you can use the RESETKEYS command to recover control of the keyboard for program editing and of the keypad for accessing setup.

Syntax
RESETKEYS

Example
RESETKEYS

REM

Usage
Enables you to add comments or reference remarks to the program code. This information is non-executable and is typically used to describe or explain the program operation. The JagBASIC preprocessor deletes all REM statements in building the executable JagBASIC program.

Syntax
REM comment

comment Text in any combination of characters.

Example
10 REM This is a comment.

RUN

Usage
Executes the current file in memory. If no program is resident in memory when RUN is executed, JagBASIC returns to the command prompt. Resumes execution at next line number after BREAK (JAGXTREME only).

Syntax
RUN ["filename.bas"]

filename.bas The name of the file that you want to execute. All open files will be closed and the new program loaded into memory and executed. If a filename is not specified, the current open program is executed.

Example
RUN "test.bas"

RZ

Usage
Initiates receiving files into the terminal's RAMDISK file system from the PC using ZMODEM protocol over serial port 1.

Syntax
RZ

Example
RZ

SAVE

Usage
Saves the current BASIC program in memory to the RAMDISK with the specified file name.

Syntax
SAVE "filename.bas"

filename.bas Name under which you want to save the current BASIC program.

Example
SAVE "file1.bas"

SHOW

Usage
SHOW displays the last line number executed in square brackets, the variable name, and its current value. SHOW is a program debug command.

Syntax
SHOW variable

Example
SHOW a$

STEP

Usage
Executing the STEP command at a breakpoint executes the next line number and stops. Pressing the **ENTER** key at a breakpoint performs the STEP function. STEP is a program debug command.

Syntax
STEP

Example
STEP

STOP

Usage
Terminates program execution and returns to the command level. STOP may be used anywhere in a program to terminate execution. When STOP is encountered, the terminal displays the message: "end pgm." A STOP command with optional line number inserts a temporary breakpoint at the line number (JAGXTREME only). JagBASIC removes the temporary breakpoint after the line executes and the execution stops at this BREAK. Only one temporary breakpoint can be used at a time.

Syntax
STOP [line number]

Example
10 INPUT A, B, C

20 PRINT A, B, C

30 STOP

STOP 200

SZ

Usage
Initiates sending files from the terminal's RAMDISK to the PC using a Zmodem file transfer over serial port 1.

Syntax
SZ ["filename"]

filename The name of the file to be transmitted. If you do not specify a file name, Zmodem transmits all files in the RAMDISK.

Example
SZ "file1.bas"

TRON, TROFF

Usage
Enables and disables tracing of program statements. TRON and TROFF can be used to help debug the program.

TRON (Trace On) enables a trace flag that prints each line number of the program as it executes. The numbers appear enclosed in brackets. The output will use the LPRINT device.

TROFF (Trace Off) disables the trace flag.

Syntax
TRON
TROFF

Example
```
10 B=10
20 FOR C=1 to 2
30 D=B +10
40 PRINT B;C;D
50 B=B + 10
60 NEXT
70 END
TRON
RUN
[10] [20] [30] [40] 1 10 20
[50] [60] [30] [40] 2 20 30
[50] [60] [70]
TROFF
```

VARS

Usage
Prints a list of all variables to the LPRINT device.

Syntax
VARS

Example

```
variable <sb>      INTEGER val: <0>
variable <sa>      INTEGER val: <0>
variable <w2>      STRING  val: <     100.00>
variable <w1>      STRING  val: <      200.2>
4 variables        90 max
```

WATCH

Usage
WATCH is a program debug command. A watchpoint is a variable that JagBASIC monitors during execution. When the program writes a value to the variable, WATCH displays the line number of the statement in square brackets, the variable name, and its new value. If you use the optional BREAK parameter, the program stops after the current line executes. Multiple statements within a line that affect the variable value will result in multiple display lines before execution stops. You can remove a watchpoint from the list with the WATCH OFF option or by clearing all watchpoints with the CLEAR or CLEAR WATCH statements. The WATCH command without any parameters displays the current list of all watchpoints.

Syntax
WATCH variable [BREAK I OFF]

Example
WATCH a$

Variable Commands

JagBASIC enables you to represent two fundamental kinds of data: strings and numbers. Number data is further divided into "types." JagBASIC has three numeric data types and one string type.

Integer (A%)—a numeric variable representing a whole number between -32768 and +32767.

Data variables defined in the program are saved in the JagBASIC interpreter until the terminal is powered down, the NEW command is issued, or a new program is loaded using the LOAD command.

Single precision (A!)—a numeric variable in 32-bit floating point notation between 3.4E-38 to 3.4E+38.

Double precision (A#)—a numeric variable in 64-bit floating point notation between 1.7E-308 to 1.7E+308.

Variable length string (A$)—a list of characters terminated by a 0. The maximum length string is 160 bytes in the JAGXTREME terminal.

Variable names of 8 characters or less make the most efficient use of memory.

JagBASIC enables you to assign descriptive names to data values, called variables. Variable names can contain up to 16 characters and must begin with a letter. Valid characters are A-Z and 0-9. Variables are case sensitive, for example A$ and a$ are different variables. The last character of the variable name specifies the data type (%, !, #, or $). The maximum number of variables is 200.

JAGXTREME terminals use a mechanism called shared data for the various program threads to share variable data. The link to shared data from JagBASIC is implemented with a unique JagBASIC function.

DEFSHR ABC,fieldname

ABC The internal reference (variable) in BASIC for a variable in shared data with a specified fieldname.

fieldname Any terminal shared data variable name as listed in Chapter 6 or 7.

Assignments to shared data appear the same as standard variables, i.e.,

ABC = SQR(XYZ!)

Shared data inputs to expressions or functions also appear the same, i.e.,

XYZ! = ATAN(ABC)

Shared data long integers are converted to double precision type in JagBASIC when reading or writing to shared data. A long integer is a four-byte (32-bit) signed number. Bit fields in shared data are converted to integers.

JagBASIC does not support using array variables as an index into an array.

JagBASIC provides a simple structure called the array to manipulate lists of data. An array is a collection of values stored in elements that are accessed by indexing into an array. It can hold only one type of variable. Arrays function as data storage and retrieval tools in memory, just as files function as data storage and retrieval tools on disk. Arrays are used as tools for organizing and processing data. An array enables you to create a set of variables with a common name. Declaring the name and type of an array and setting the number of elements and their arrangement in the array is referred to as defining, or dimensioning, the array. Arrays may have up to three dimensions.

JagBASIC provides several data commands.

Command	Usage
COMMON	Defines global variables that can be shared between chained programs.
DATA	Specifies values to be read by READ statements.
DEFSHR	Allows a program to access the terminal shared database.
DIM	Declares an array, where subscripts are the dimensions of the array.
LET	Assigns the value of an expression to a variable.
OPTION BASE	Declares the minimum value for array subscripts.
READ	Reads values from a DATA statement and assigns them to variables.
RESTORE	Allows DATA statements to be reread from a specified line.
SWAP	Exchanges the values of two variables that are variables of the same data type.

TIPS

The LET command is optional and its use is not recommended. The following two statements are equivalent: LET X=1 and X=1.

COMMON

Usage
Defines global variables that can be shared between chained programs.

By default variable names in a program module are available only in that program module. COMMON extends the scope of listed variables to other chained programs.

Syntax
COMMON variablelist

variablelist One or more variables to be shared.

Example
```
COMMON a$,pi#
```

DATA

Usage
Specifies values to be read by READ statements. DATA statements contain lists of values separated by commas. The first READ statement in a program reads the first value in the DATA list. The second READ statement reads the second value in the DATA list, and so on. JagBASIC tracks the next value to be read.

Syntax
DATA constant[,constant]...

constant One or more numeric or string constants specifying the data to be read. String constants containing commas, colons, or leading or trailing spaces are enclosed in quotation marks ("").

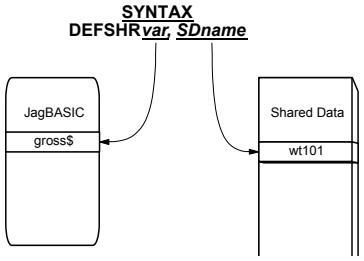
Example
```
10  DIM item$(5), number(5,3)
20  FOR k% = 1 to 5
30  READ item$(k%)
33  FOR j% = 1 to 3
35  READ number(k%,j%)
36  NEXT j%
40  NEXT k%
45  FOR j% = 1 to 3
60  FOR k% = 1 to 5
70  LPRINT item$(k%), number(k%,j%)
80  NEXT k%
85  NEXT j%
90  DATA hammers,4,5,6,umbrellas,2,3,4,wood_stoves,1,2
100  DATA bags_of_salt,4,5,6,needle_nose_pliers,2,3,4
110 END
```

Output:	hammers	4
	umbrellas	2
	wood_stoves	1
	bags_of_salt	4
	needle_nose_pliers	2
	hammers	5
	umbrellas	3
	wood_stoves	2
	bags_of_salt	5
	needle_nose_pliers	3
	hammers	6
	umbrellas	4
	wood_stoves	3
	bags_of_salt	6
	needle_nose_pliers	4

DEFSHR

Usage

Allows a program to access the terminal shared database. Any read or write to the variable name automatically refers to the associated field within the shared database. JagBASIC automatically determines the variable type from the shared file name. The shared file name overrides the variable name suffix.



Once the DEFSHR command is executed for a variable, the shared data variable may be read or written using JagBASIC's variable name for it. The variable type (string, float, integer) must match the shared data type; otherwise a syntax error is indicated. No type conversion is performed.

Syntax

DEFSHR variablename,sharedfilename

variablename The name of the variable.

sharedfilename The name of the shared data file.

Example

This program displays the gross weight of the scale not selected in the lower terminal display using the "print" command. The "a" and "b" keys on the terminal keyboard enable you to switch between Scale A and Scale B.

```
1   REM w1=gross weight Scale A, w2=gross weight scale B.
2   REM sa is the discrete event to select scale A.
3   REM sb is the discrete event to select scale B.
4   REM Display the gross weight of the scale not selected in
5   REM the lower terminal display using the "print" command.
6   REM Switch between Scale A and Scale B using the
```

```
7    REM "a" and "b" keys on the terminal keyboard.
10   DEFSHR w1,wt101
20   DEFSHR w2,wt201
30   DEFSHR sa,t_6c0
40   DEFSHR sb,t_6c1
50   sa=1
60   PRINT " wa= ";w2
70   IF INKEY$ = "b" THEN GOTO 100
80   GOTO 60
100 sb=1
110 PRINT " wb= ";w1
120 IF INKEY$ = "a" THEN GOTO 50
130 GOTO 110
140 END
```

DEFSHR Arrays

You may use single dimension or multidimensional arrays of DEFSHRs. JagBASIC allows you to setup an array of DEFSHRs so that you can index into an array of shared data variables. This feature reduces the amount of JagBASIC code needed for accessing multiple scales, setpoints, discrete input variables, discrete output variables, and literals. You must use the dimension statement to define the array type and array size. Then, you use the DEFSHR statement to assign a shared data variable to each element of the array. The type of each shared data variable must be the same type as the array.

Example 1

```
5    REM ARRAY OF SHARED DATA LITERALS
10   DIM a$(5)
20   DEFSHR a$(1),lit01
30   DEFSHR a$(2),lit02
40   DEFSHR a$(3),lit03
50   DEFSHR a$(4),lit04
60   DEFSHR a$(5),lit05
70   FOR i% = 1 to 5
80   LPRINT "literal";i%;" = ";a$(i%)
90   NEXT i%
100 END
```

Example 2

```
5   REM ARRAY OF SETPOINT COINCIDENCE VALUES
10 DIM setpoint#(4)
20 DEFSHR setpoint#(1),sp105
30 DEFSHR setpoint#(2),sp305
40 DEFSHR setpoint#(3),sp505
```

```
50 DEFSHR setpoint#(4),sp705
60 FOR i% = 1 to 4
70 setpoint#(i%)=2.0*i%
80 NEXT i%
90 END
```

Example 3

```
5    REM ARRAY OF DISCRETE OUTPUTS
10   DIM do%(12)
20   DEFSHR do%(1),p_500
30   DEFSHR do%(2),p_501
40   DEFSHR do%(3),p_502
50   DEFSHR do%(4),p_503
60   DEFSHR do%(5),p_508
70   DEFSHR do%(6),p_509
80   DEFSHR do%(7),p_50a
90   DEFSHR do%(8),p_50b
100 DEFSHR do%(9),p_50c
110 DEFSHR do%(10),p_50d
120 DEFSHR do%(11),p_50e
130 DEFSHR do%(12),p_50f
140 FOR j% = 1 to 10
150 FOR i% = 1 to 12
160 DO%(i%)=1
170 SLEEP 1000
180 DO%(i%)=0
190 NEXT i%
200 NEXT j%
```

DEFSHR Links to Remote Shared Data

JagBASIC programs can access shared data variables located in remote JAGXTREME terminals in a cluster. The node location and name of the remote data variable is specified in a DEFSHR statement. When there is no node specified in the DEFSHR command, JagBASIC assumes that the request is for the local node.

DEFSHR a$,j2/wt101 creates a remote link to gross weight shared data variable wt101 in terminal node j2.

DEFSHR b$,wt101 creates a local link to gross weight shared data variable wt101.

Once the link has been established, use the normal JagBASIC syntax to access the remote variable. The program should take into account that the remote terminal may not be online when you attempt to access it. Use an ON ERROR GOTO or ON ERROR GOSUB statement to handle these offline errors.

The program should also take into account that there is a short time delay when it accesses the remote shared data variable. When the BASIC application uses the same remote variable more than once in a series of calculations, it should access it only once and store the value in a local BASIC variable. Then, the BASIC application can use the local BASIC variable in the subsequent calculations. This procedure will streamline the execution speed of the program.

Example

```
10   REM SUM GROSS WEIGHT IN A CLUSTER
20   ON ERROR GOSUB 1000
30   DIM w#(4,2)
40   DEFSHR w#(1,1),j1/wt110
50   DEFSHR w#(1,2),j1/wt210
60   DEFSHR w#(2,1),j2/wt110
70   DEFSHR w#(2,2),j2/wt210
80   DEFSHR w#(3,1),j3/wt110
90   DEFSHR w#(3,2),j3/wt210
100  DEFSHR w#(4,1),j4/wt110
110  DEFSHR w#(4,2),j4/wt210
120  SUM#=0
130  FOR i% = 1 to 4
140  FOR i% = 1 to 2
150  sum#=sum#+w#(i%,j%)
160  NEXT j%
170  NEXT i%
180  PRINT using "total_+#####.##";sum#
190  TPRINT using "total_+#####.##";sum#
200  GOTO 120
1000 IF err()<>32 or erl()<>150 then end
1010 PRINT "JAGXTREME ";i%;" offline"
1020 IF INKEY$ = "" then GOTO 1020
1030 RETURN
```

DIM

Array variables "can not" be used as part of any serial input statement,.

Example of illegal operation:

```
10 open "com1:xpr len10 trm13
 tmo100" for input as #1
20 Input #1, data$(x)
```

Usage

Declares the name, size and type of an array and allocates storage for it. An array is a variable containing a series of values that are referred to with one name. The number in parentheses following the array name defines the number of individual variables in the array. A JagBASIC array can have up to three dimensions.

Syntax

DIM variable[(subscripts)] [,variable[(subscripts)]]

variable Name of an array.

subscript Used in conjunction with variable; defines dimensions of array.

Example

```
10  DIM item$(5), number(5,3)
20  FOR k% = 1 to 5
30  READ item$(k%)
33  FOR j% = 1 to 3
35  READ number(k%,j%)
36  NEXT j%
40  NEXT k%
45  FOR j% = 1 to 3
50  PRINT "You have these items:"
60  FOR k% = 1 to 5
70  PRINT item$(k%), number(k%,j%)
80  NEXT k%
85  NEXT j%
90  DATA  hammers,4,5,6,umbrellas,2,3,4
95  DATA  wood_stoves,1,2,3
100 DATA bags_of_salt,4,5,6,pliers,2,3,4
110 END
```

LET

Usage

Assigns the value of an expression to a variable. Use of the keyword LET is optional. This command is used to initialize variables or to change their current value. The command word LET is optional and its use is not recommended.

Syntax

[LET] variable=expression

variable	The variable name.
expression	The value that you want to assign to the variable name.

Example 1

```
10  LET A$ = "JAGXTREME"
```

Example 2

```
20  B$ = "JagBASIC"
```

OPTION BASE

Usage

Declares the minimum value for array subscripts. Subscripts are the numbers which can be used to access the elements of an array. OPTION BASE gives an error if the base value is changed. The default subscript base is 1.

Syntax
OPTION BASE {0 | 1}

0 Sets the lowest value any array subscript can have to 0.

1 Sets the lowest value any array subscript can have to 1. This is the default setting.

Example 1
OPTION BASE 1

Example 2
OPTION BASE 0

READ

Usage
Reads values from a DATA statement and assigns them to variables. Values are always read in the order in which they appear in the DATA statements.

Syntax
READ variablelist

variablelist One or more variables, separated by commas.

Example
```
70  DIM A(10)
80  FOR I=1 TO 10
90  READ A(I)
100 NEXT I
110 DATA 3.10,5.20,6.10,7.20,8.10
120 DATA 5.30,6.30,7.30,8.30,9.30
```

RESTORE

Usage
Allows DATA statements to be reread from a specified line. Enables a program to read data selectively based on a particular condition.

Syntax
RESTORE [line]

line The line number of a DATA statement. If line is omitted, the next READ accesses the first item in the first DATA statement.

Example
```
10  READ A,B,C
20  RESTORE
30  READ D,E,F
40  DATA 57,58,59
```

SWAP

Usage
Exchanges the values of two variables, if the variables are the same data type.

Syntax
SWAP variable1, variable2

variable1	One of the variables whose value you want to exchange.
variable2	One of the variables whose value you want to exchange.

Example

```
10  a% = 1: b% = 2
20  PRINT "Before: "; a%, b%
30  SWAP a%, b%
40  PRINT "After: "; a%, b%
```

Output: Before 1,2
After 2,1

Flow Control and Operator Commands

Few programs run straight through the program code from the first statement to the last in sequence. Usually, you need to branch to a different piece of code or repeat a section multiple times. Identical tasks that are used in several places can also be made into a subroutine to save code space. This section details how JagBASIC allows you to control the sequence of program execution.

Branching directs control of the code away from the next sequential step. JagBASIC has two commands that can be used to perform branching: GOTO and GOSUB .... RETURN.

GOTO causes the program to jump to a different execution point and continue sequencing from the line number indicated until the program ends or encounters another "branching" command.

GOSUB .... RETURN causes the program to jump to a different execution point and then return to the statement following the original branching point once the RETURN statement is reached.

IF condition THEN line executes an implied GOSUB call to the appropriate line depending on the specified condition. The program jumps to a different execution point and then returns to the statement following the original branching point once the RETURN statement is reached.

Looping executes the same sequence of statements more than one time. JagBASIC has two loop commands: FOR .... NEXT and WHILE .... WEND.

The FOR .... NEXT loop is repeated a fixed number of times as determined in the statement's first line.

The WHILE .... WEND loop is repeated until a condition has been met.

Decision and operator commands enable programs to change processing based on certain criteria. JagBASIC's fundamental criteria determining statement is the IF .... THEN command.

IF …. THEN, used in conjunction with the logical operators, AND, OR, and XOR, enables you to establish specific conditions which must be met in order for a resulting action to occur.

This section discusses the following flow control and operator commands:

Command	Usage
AND	A logical operator in a decision statement which establishes two sets of criteria to be met.
CHAIN	Dynamically loads another program file for execution and begins executing the program.
CHAINCALL	Operates same as CHAIN command except it remembers the current program name and line number of the program initiating chaining.
CHAINRET	Operates same as CHAIN command except it returns control from the chained program to the chaining program of next line after the CHAINCALL.
FOR…NEXT	Repeats a section of the program the specified number of times.
GOSUB	Branches to a specified line number with intent to return to the next line.
GOTO	Branches unconditionally to the specified line number.
IF…THEN	Executes the sub-statement depending on specified conditions.
OR	Used as a logical operator in a decision statement to establish two possible conditions, of which only one needs to be met.
RESETJAG	Resets the terminal by forcing its execution through power-up.
RESTART	Clears the JagBASIC execution stacks and sends program control to the first line of the current program.
RETURN	Used in conjunction with GOSUB, indicates that the subroutine is complete.
SWITCHSUB	Branches to a line number specified by the value of a variable with intent to return to the next line.
SWITCHTO	Branches unconditionally to the line number specified by the value of a variable.
WHILE…WEND	Repeats a section of the program until a specified logical condition is true.
XOR	Used as a logical operator in a decision statement to establish two possible conditions, only one of which can be met.

TIPS

JagBASIC does not support commands for breaking out of loops other than their normal exit point. For this reason programmers may try to branch out of loops. Do not jump from inside a loop to outside the range of the loop. Always take the normal return from a GOSUB command. JagBASIC supports nine levels of nesting for GOSUB, FOR-NEXT, and WHILE-WEND. If you branch out of these structures, the nesting level does not get reset. Eventually, an overflow error will occur.

AND

The AND operator has a lower precedence than assignment operators. Use parentheses around the operation to assign its value to a variable.

Usage

Used as a logical operator in a decision statement to establish two sets of criteria, both of which must be met. AND can also be used as a bitwise operator between two integer expressions.

Syntax
IF condition1 AND condition2 THEN result.

condition1 First condition for decision.

condition2 Second condition for decision.

result Result that will occur if both conditions are met.

Example 1
30 IF A>75 AND B<20 THEN 5000

Example 2
50 A% = (B% AND 1)

CHAIN

For the most efficient memory utilization, start execution with the largest program and chain the smaller programs.

Usage
Allows you to program a large application by enabling you to split the application into smaller program modules. CHAIN loads another program and transfers control from the current program to another BASIC program. Variables identified as common variables are accessible by the chained JagBASIC program. CHAIN commands must be placed in the top level of the JagBASIC program, not within a GOSUB, IF-THEN-ELSE, WHILE-WEND, or FOR-NEXT loop.

Syntax
CHAIN "filename.bas"

filename.bas The name of the program in the terminal RAMDISK directory to which you want to transfer the current program's controls and variables.

Example
CHAIN "test.bas"

CHAINCALL

Usage
The CHAINCALL command as operates the same as a CHAIN command except that it remembers the current program name and line number of the program that is initiating the chaining. After issuing a CHAINCALL, you must execute a CHAINRET before issuing another CHAINCALL.

Syntax
CHAINCALL "filename.bas"

Filename.bas The name of the program in the RAMDISK directory you want to transfer to the current program's controls and variable with the intent to return.

Example

```
5  REM Program CHAIN_X.BAS
10 LPRINT "The square root of "; 9; " is "; sqr(9)
20 LPRINT "The square root of "; 6; " is "; sqr(6)
30 LPRINT "The difference is "; sqr(9) - sqr(6)
40 LET num = 5.0 / 3
50 LPRINT "5 divided by 3 is ";  num
60 LPRINT "5 divided by 2 is ";  5.0 / 2
70 PRINT "hello"
```

```
80 CHAINCALL "chain_y.bas"
90 PRINT "bye bye"
100 SLEEP 300
120 GOTO 10
---------------------------------
REM Program CHAIN_Y.BAS
LPRINT "This is Y"
LPRINT "Time to return"
30  CHAINRET
```

CHAINRET

Usage

The CHAINRET command operates the same as a CHAIN command except that it returns control from the chained program to the chaining program at next line after the CHAINCALL.

Syntax

CHAINRET

Example

```
5  REM Program CHAIN_Y.BAS
10 DIM item$(5), number(5,3)
20 FOR k% = 1 to 5
30 READ item$(k%)
33 FOR j% = 1 to 3
35 READ number(k%,j%)
36 NEXT j%
40 NEXT k%
45 FOR j% = 1 to 3
50 LPRINT "You have these items:"
60 FOR k% = 1 to 5
70 LPRINT item$(k%), number(k%,j%)
80 NEXT k%
85 NEXT j%
90 DATA hammers,4,5,6,umbrellas,2,3,4,wood_stoves,1,2,3
100 DATA bags_of_salt,4,5,6,pliers,2,3,4
110 LPRINT "I am here"
120 SLEEP 300
140 CHAINRET
```

FOR .... NEXT

Usage
Repeats the block of statements between the keywords FOR and NEXT the specified number of times.

Syntax
FOR counter = start TO end [STEP increment]

*

*

*

*

NEXT counter

counter A numeric variable used as the loop counter.

start The initial value of the counter.

end The final value of the counter.

increment The amount the counter changes each time through the loop. A fractional value is not allowed. If STEP is not specified, JagBASIC assumes a value of 1.

Example 1

```
100 FOR j% = 1 TO 15
110 PRINT j%
120 SLEEP 1000
130 NEXT j%
```

Example 2

```
100 FOR a% = 1 TO 100 STEP 10
110 PRINT a%
120 NEXT a%
```

GOSUB

Usage
Branches to a subroutine. Used in conjunction with RETURN.

Syntax
GOSUB line

*

*

line *

*

RETURN

line The line number of the subroutine to branch to in the program.

Example

```
10 FOR b% = 1 TO 20
20 GOSUB 50
30 NEXT b%
40 END
50 REM Print Subroutine
60 PRINT "b%= ",b%
70 RETURN
```

GOTO

Usage
Branches unconditionally to a specified line.

Syntax

```
GOTO line
*
*
*
line *
```

line The line number to branch to in the program.

Example

```
10 IF INKEY$="E" then GOTO 50
20 GOTO 10
50 END
```

IF .... THEN

Usage
Executes the sub-statement depending on specified conditions. The entire IF statement must be contained on one line. The condition is any expression that can be evaluated as true or false. You can have multiple statements in a THEN or ELSE clause as long as the entire statement is contained on one line. The IF condition THEN line statement executes an implied GOSUB call to the appropriate line depending on the specified conditions. Be sure to execute a RETURN from this implied GOSUB.

Syntax

```
IF condition THEN statement [ELSE statement]
IF condition THEN GOTO linenumber [ELSE statement]
IF condition THEN line [ELSE line]
```

Example 1

```
10 INPUT "SELECTION ? ", i%
20 IF i% = 1 THEN PRINT "OK" ELSE GOTO 50
30 GOTO 10
50 END
```

Example 2

```
10 FOR i% = 1 to 10
20  IF i% < 7 THEN 100 ELSE 120
30 NEXT i%
40 END
100 PRINT " You lose."
115 RETURN
120 PRINT " You win."
130 RETURN
```

Example 3

```
NextKey:
m$=inkey$
IF m$=chr$(2) THEN x%=2: GOTO Escape
IF m$=chr$(3) THEN x%=3: GOTO Memory ELSE x%=0: GOTO NextKey
Escape:
PRINT "Escape"; x%: GOTO NextKey
Memory:
PRINT "Memory"; x%: GOTO NextKey
```

OR

The OR operator has a lower precedence than assignment operators. Use parentheses around the operation to assign its value to a variable.

Usage

Used as a logical operator in a decision statement to establish two possible conditions, of which only one needs to be met. OR can also be used as a bitwise operator between two integer expressions.

Syntax

IF condition1 OR condition2 THEN result

condition1	First condition for decision.
condition2	Second condition for decision.
result	Result that will occur if either condition is met.

The first example is an implied GOSUB and requires a RETURN statement later in the program.

Example 1

```
30 IF A>75 OR B<20 THEN 5000
```

Example 2

```
30 IF A>75 OR B<20 THEN GOTO 5000
```

Example 3

```
10 B% = (A% ORC%)
```

RESETJAG

Usage
The RESETJAG command re-initializes the terminal by forcing execution through the power-up cycle.

Syntax
RESETJAG

Example
RESETJAG

RESTART

Usage
Clears the JagBASIC execution stacks and sends program control to the first line of the current program. This command does not affect the BASIC variables.

Syntax
RESTART

Example
RESTART

RETURN

Usage
Branches back to the line following a GOSUB statement.

Syntax
RETURN

Example
```
10 GOSUB 1000
*
1000 LPRINT "Hello"
1010 RETURN
```

SWITCHSUB

Usage
Performs a GOSUB call to the line specified in the variable.

Syntax
SWITCHSUB lineNumber%

lineNumber% is a variable containing the line number to which control is transferred.

Example 1
```
110     IF a%=1 THEN j%=1000 ELSE j%=2000
120     switchsub j%
130     REM Main Loop
140     a%=0
150     END
1000    LPRINT "Test complete"
1010    RETURN
```

```
2000        LPRINT "Select Test"
2010        RETURN
```

Example 2
Code before running through JagBASIC preprocessor. The JagBASIC preprocessor resolves labels that are identified by xx to line numbers.

```
REM table initialization
DIM cmd% (18)
j%=1
NextCmd:
Read cmd%(j%)
if j%<18 then j%=j%+1; GOTO NextCmd
*
*
REM call subroutine to process command
ProcessCommand:
Input "^command"; j%
If j%<0 or j%>18 then GOTO ProcessCommand
switchsub j%
*
SetDischargeCycle:
*
CaptureTare:
*
*
REM command state table
DATA xx SetDichargeCycle, xx CloseGates, xx CloseGate
DATA xx WaitForWHGateClose, xx CloseGate, xx WaitUGClose
DATA xx WaitSettlingTimer, xx WaitNoMotion, xx NoMotion
DATA xx CaptureTare, xx RecordDraftComplete, xx CheckUpstreamPreact
DATA xx SetFill cycle, 0,0,0,0,0
```

SWITCHTO

Usage
Performs a GOTO operation to the line specified in the variable.

Syntax
Switchto lineNumber%

lineNumber% is a variable value that specifies the location to GOTO.

Example

```
100 IF a%=1 THEN j%=1000
110 IF a%=2 THEN j%=1100
120 IF a%=3 THEN j%=1200
500 SWITCHTO j%
1000 LPRINT"Test 1"
1010 a%=0
1020 GOTO 2000
1100 a%=1
1120 GOTO 2000
1200 LPRINT "Test 3"
1210 a%=2
2000 *
```

WHILE...WEND

Usage
Executes a series of statements as long as a specified condition is true. If the condition is false when the WHILE statement is first encountered, the loop is bypassed and not executed.

Syntax
WHILE condition

.

.
WEND

Example

```
10  years=0
20  money=10000
30  start=money
40  interest=8.5/100
50  WHILE money <= 2*start
60  PRINT years,money
70  years = years+1
80  money = money+(interest*money)
90  WEND
```

```
100 PRINT "In "; years ; " years, you'll have $"; money
110 END
```

XOR

The XOR operator has a lower precedence than assignment operators. Use parentheses around the operation to assign its value to a variable.

Usage

Used as a logical operator in a decision statement to establish two possible conditions, only one of which can be met. Used to guarantee that only one variable is true, preventing conflicting options from being true. XOR can be used as a bitwise operator between two integer expressions. The XOR operator has a lower precedence than assignment operators. Use parentheses around the operation to assign its value to a variable.

Syntax

IF condition1 XOR condition2 THEN result

condition1	First condition for decision.
condition2	Second condition for decision.
result	Result that will occur if only one conditions is met.

Example 1

This example is an implied GOSUB statement.

```
30 IF A>75 XOR B<20 THEN 5000
```

Example 2

```
100 x%=(4 XOR A%)
```

Precedence of Operators

JagBASIC's order of operations has a predefined precedence when evaluating expressions. The following numeric and conditional operators are in precedence order.

^	Exponent
*	Multiply
/	Divide
\	Integer Divide
MOD	Modulus
+	Add
-	Subtract
=	Equals
=	Assign
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than Or Equal
>=	Less Than Or Equal
=>	Greater Than Or Equal
NOT	Not
AND	And
OR	Or
XOR	Exclusive Or

For example,

60 B=3+4*5

70 PRINT B

Output: 23

AND, OR, and XOR have lower precedence than an assignment operator. Therefore, if you need to assign the results of an AND, OR, and XOR operation to a variable, you must put parentheses around the operation.

Math Commands

JagBASIC provides numerous advanced mathematical commands. Using the commands listed in this section, you can perform the following types of mathematical functions:

Trigonometric commands ATN, COS, SIN, and TAN return the arctangent, sine, cosine, and tangent. The angle values are expressed in radians. To convert to degrees, multiply the number of radians by (180/π) or approximately 57.3°.

Logarithmic and Exponential commands return the natural logarithm and its complement. Natural logarithms are based on e (approximately 2.718282.)

Conversion commands convert numbers from one type to another. These commands enable you to convert a number from its existing format to the format expected by the function or subroutine. Conversion is implied by the variable's data type. For example, a#=1 automatically converts the integer 1 to a double precision floating point number.

Rounding and Truncating commands round and truncate numbers.

Random Number commands generate random numbers.

Arithmetic Operations commands perform operations such as finding a number's absolute value, determining its sign, and finding its square root.

This section discusses the following JagBASIC mathematical commands:

Command	Usage
ABS()	Returns the absolute value of a number.
ATN()	Returns the arctangent of specified numeric expression in radians.
CINT	Rounds a numeric expression to the closest integer.
COS()	Returns the cosine of a specified angle expressed in radians.
CSNG()	Converts a numeric expression to a single-precision value.
EXP()	Returns e raised to a specified power, where e is the base of natural logarithms.
INT()	Returns the largest integer less than or equal to a numeric expression.
LOG ()	Returns the natural logarithm of a numeric expression.
RANDOMIZE	Initializes the random-number generator.
RND ()	Returns a single-precision random number between 0 and 1.
SGN ()	Returns a value indicating the sign of a numeric expression.
SIN()	Returns the sine of a specified angle expressed in radians.
SQR()	Returns the square root of a numeric expression.
TAN()	Returns the tangent of a specified angle expressed in radians.

TIPS

If you specify nonnumeric values with any of the mathematical commands, you will receive a type mismatch error message.

ABS()

Usage
Returns the absolute value of a number. The absolute value of a number is the magnitude of the number without regard to sign. Absolute values are always positive numbers.

Syntax
ABS(numeric-expression)

numeric-expression Any numeric expression.

Example
10 PRINT ABS(45.5-100)

Output: 54.5

ATN()

Usage
Returns the arctangent of a specified numeric expression in radians. The arctangent is the angle whose tangent is equal to the specified value.

Syntax
ATN(numeric-expression)

numeric-expression Any numeric expression expressed in radians.

Example
10 LPRINT ATN(.75), ATN(.9)

Output (in radians): 0.6435011 0.7328151

CINT()

Usage
Rounds a numeric expression to the closest integer. The numeric expression can be any number in the range -32,768 through 32,767.

For positive numbers

If the numeric expression contains a fractional part that is less than 0.5, CINT rounds to the next lower integer.

If the numeric expression contains a fractional part that is greater than or equal to 0.5, CINT rounds to the next higher integer.

For negative numbers

If the numeric expression contains a fractional part that is less than 0.5, CINT rounds to the next higher integer.

If the numeric expression contains a fractional part that is greater than or equal to 0.5, CINT rounds to the next lower integer.

Syntax
CINT(numeric-expression)

numeric-expression Any numeric expression.

Example
10 PRINT CINT(12.49), CINT(12.51), CINT(12.50), CINT(-12.49)

Output: 12 13 12 -12

COS()

Usage
Returns the cosine of a specified angle expressed in radians.

Syntax
COS(angle)

angle Angle expressed in radians.

Example
```
40 pi#=3.141592654
50 LPRINT COS(180*pi#/180)
```
Output: -1

CSNG()

A single precision numeric variable represents a number of seven or fewer digits plus an exponent.

A double precision numeric variable represents a number of eight or more digits plus an exponent.

Single-precision and double-precision are also referred to as floating point variables.

Usage
Converts a numeric expression to a single-precision value.

numeric-expression Any numeric expression.

Syntax
CSNG(numeric-expression)

Example
```
PRINT CSNG(975.342151523497)
```
Output: 975.342152

EXP()

Usage
Returns e raised to a specified power. The natural logarithm base, e, has a value of approximately 2.71828. The natural logarithm of a number is the power to which the base e must be raised to obtain the number. EXP is the inverse function of the natural log function.

Syntax
EXP(numeric-expression)

numeric-expression Any numeric expression.

Example
```
PRINT EXP(0), EXP(1)
```
Output: 1 2.718282

INT()

Usage
Returns the integer portion of a specified numeric expression.

For positive numbers, the fractional part of the numeric expression is truncated, that is cut-off.

For negative numbers, the next lower integer is returned.

Rounding does not occur with this command.

Syntax
INT(numeric-expression)

numeric-expression Any numeric expression.

Example
10 PRINT INT(12.54), INT(-99.4)

Output: 12 -100

LOG()

Usage
Returns the natural logarithm of a numeric expression. Natural logarithms are based on *e,* which is approximately 2.718282. The natural logarithm of a number is the power to which the base *e* must be raised to obtain the number.

Syntax
LOG(numeric-expression)

numeric-expression Any positive numeric expression.

Example
10 PRINT LOG(5), LOG(EXP(1))

Output: 0.69897 1

RANDOMIZE, RND ()

Usage
RANDOMIZE specifies a particular initial value or seed value for the random number generator. This seed value is used in specifying the random-number series to be used when the program calls the RND function.

RND returns a single-precision random number between 0 and 1. The same sequence of random numbers is generated each time the program runs unless the RANDOMIZE statement was used to specify a different sequence.

RND returns a pseudorandom number which is generated from the seed value using a formula designed to produce numbers that have no pattern or order and appear to be random. Each seed actually creates a fixed sequence of numbers. RANDOMIZE enables you to change the seed value and the sequence generated.

Syntax
RANDOMIZE [seed%]

RND[(n#)]

seed% A number used to initialize the random-number generator.

n# A value that sets how RND generates the next random number.

Example

```
10 RANDOMIZE
20 FOR game% = 1 to 10
30  die1 = INT(6*RND + 1)
40  die2 = INT(6*RND + 1)
50  dice = die1 + die2
60  PRINT dice;
70  IF dice < 7 THEN GOSUB 100 ELSE GOSUB 120
80 NEXT game%
90 GOTO 150
100 PRINT " You lose."
115 RETURN
120 PRINT " You win."
130 RETURN
150 END
```

SGN ()

Usage

Returns a value indicating the sign of a numeric expression. Used to test whether a value is negative, positive, or zero.

Syntax

SGN(numeric expression returns)

1	The expression is positive.
0	The expression is zero.
-1	The expression is negative.

Example

```
10 PRINT SGN(12), SGN(-15), SGN(0)
```

Output: 1 -1 0

SIN()

Usage

Returns the sine of a specified angle expressed in radians.

Syntax

SIN(angle)

angle	Angle expressed in radians.

Example

```
10 pi#=3.141592654
20 LPRINT SIN(90*pi#/180)
```

Output: 1

SQR()

Usage
Returns the square root of a positive numeric expression.

Syntax
SQR(numeric-expression)

numeric-expression Any numeric expression.

Example
10 PRINT SQR(25), SQR(2)

Output: 5 1.414214

TAN()

Usage
Returns the tangent of a specified angle expressed in radians.

Syntax
TAN(angle)

angle Angle expressed in radians.

Example
10 pi#=3.141592654

20 LPRINT TAN(45*pi#/180)

Output: 1

String Commands

JagBASIC enables you to form many string expressions. A string is simply a variable length series of character values. Each byte in a string expression is treated in one of two ways:

As an ASCII character with a value in the range 1 to 127. The ASCII character set includes uppercase and lowercase letters, numbers, punctuation marks, mathematical symbols, and printer control characters.

As an extended character in the range 128 through 255.

Strings are terminated by a 0 (null). The maximum length of a string is 80 characters.

To define a string variable, select a name that describes the string's contents, such as name$ for the name on a mailing label. The dollar sign ($) suffix means that the variable holds string data. Use an equal sign (=) followed by a string expression to assign a value to the string. A string expression can be as simple as a single variable name or as complex as a combination of string literals, variables, functions, and the plus sign.

Expression	Comment
"Tom and Harry"	Single literal
Name$	Single variable
RIGHT$(Name$,5)	String function
"Smith" + LastName$	Combination expression

JagBASIC's string commands enable you to:

- Extract part of a string.
- Convert decimal numbers (base 10) to hexadecimal (base 16) or octal (base 8) strings.
- Convert a character to ASCII code and the reverse.
- Create "field" strings, which are used to format and arrange output.
- Count characters in a string or the number of bytes required to store a variable.
- Display the string representation of a number.
- Locate one string within another string.
- Interpret the string entered by the user as though it were a number.
- Insert a string into another string.
- Convert a string to upper case or lower case
- Trim spaces from the beginning or end of a string

This section discusses the following string commands.

Command	Usage
ASC()	Returns the ASCII or extended code value for the first character in a string expression.
CHR$()	Returns the single-character string corresponding to the specified ASCII code.
HEX$()	Returns a string containing the hexadecimal value of a number.
INSTR ()	Returns the position of the first occurrence of a string in another string.
LCASE$	Converts a string to a lower case.
LEFT$()	Returns a specified number of leftmost characters in a string.
LEN()	Returns the number of characters in a string or the number of bytes required to store a variable.
LTRIM$ ()	Removes spaces from the beginning of a string.
MID$()	Returns part of a string.
MSET$()	Inserts one string into another string, overwriting the existing characters.
PADC$ ()	Add pad characters to beginning and end of a string.
PADL$ ()	Adds pad characters to the beginning of a string.
PADR$ ()	Adds pad characters to end of a string
OCT$()	Returns an octal string representation of a number.
RIGHT$()	Returns a specified number of rightmost characters in a string.
RTRIM$ ()	Removes spaces from the end of a string.
SPACE()	Returns a string of spaces.
STR$ ()	Returns a string representation of a number.
STRING$()	Returns a string of a specified length made up of a repeating character.
UCASE$ ()	Converts a string to upper case.
VAL()	Converts a string representation of a number to a number.

ASC()

Usage
Returns the ASCII or extended code value of the first character in the specified string expression.

Syntax
ASC(stringexpression$)

stringexpression$ Any string expression.

Example
10 PRINT ASC("Quiet")

Output: 81

The ASCII value of a capital Q is 81.

CHR$()

Usage
Returns the single-character string corresponding to the specified ASCII code. Used for characters not easily entered on the keyboard and placed in a string, such as most control characters and graphic characters. The CHR$ commands can generate all 255 characters of the ASCII and extended character sets.

Syntax
CHR$(ascii-code%)

ascii-code% ASCII or extended code of the desired character in the range of 1-255.

Example
20 PRINT CHR$(65)

Output: A

HEX$()

Usage
Converts a decimal number (base 10) to a hexadecimal number (base 16).

Syntax
HEX$(numeric-expression)

numeric expression Any numeric expression.

Example
10 INPUT x

20 a$ = HEX$(x)

30 PRINT x; "decimal is "; a$; " hexadecimal"

INSTR ()

Usage
Returns the position of the first occurrence of a string in another string. Used for searching text in database fields or for validating user input.

Syntax
INSTR(string1$,string2$)

string1$	String expression being searched.
string2$	String expression that you want to locate.

Example

```
10 DIM prglst$(5)
20 prglst$(1)="abcdefgh"
30 prg$="bcd"
40 PRINT INSTR(prglst$(1),prg$)
```

Output: 2

LEFT$()

Usage
Returns the specified number of leftmost characters in a string. If you specify a number of characters greater than or equal to the string's length, the entire string is returned.

Syntax
LEFT$(stringexpression$,n%)

stringexpression$	Any string expression.
n%	Number of characters to return. Range is 0 to 80.

Example

```
10  a$ = "JAGXTREME BASIC"
20  PRINT LEFT$(a$, 9)
```

Output: JAGXTREME

LEN()

Usage
Returns the number of characters in a string or the number of bytes required to store a variable. Used to obtain the length of a string. If a zero is returned, the string is empty.

Syntax
LEN(stringexpression$)

stringexpression$ Any string expression.

Example
```
10 A$ = "ABC"
20 WHILE LEN(A$) < 8
50  A$ = A$ + "C"
60  LPRINT  A$;" HAS LENGTH "; LEN(A$)
70 WEND
80 END
```
Output: ABCCCCCC has length 8

LTRIM$ ()

Usage
Removes the spaces from the beginning of a string.

Syntax
LTRIM$ (stringexpression$)

stringexpression$ Any string expression.

Example
```
10 a$ =  "     12345"
20 b$ = LTRIM$(a$)
```
Result: b$="12345"

MID$()

Usage
Returns part of a string. The part of the string returned begins at the specified position and contains the given number of characters. If the starting position is greater than the length of the string, a null string is returned. If the number of characters to return is greater than the length of the string, the entire string is returned.

Syntax
MID$(stringexpr$,start%[,length%])

stringexpr$ Any string expression.

start% The starting character position to read.

length% The number of characters to read.

Example
```
10  a$ = "Where is Cambridge?"
20  PRINT MID$(a$, 10, 10)
```
Output: Cambridge?

MSET$()

Usage
Inserts one string into another string at a specified position. Overwrites the existing characters so that the length of the string remains the same.

Syntax
MSET$() (string1$, string2$, position%)

string1$	string to be changed
string2$	string to insert
position%	Number of character to insert string after

Example

```
5 a$="123456789"
10 b$="abc"
15 a$=MSET$(a$,b$,3)
20 LPRINT "a$"=";a$
```

Output: a$=123abc789

OCT$()

Usage
Converts a number to an octal string.

Syntax
OCT$(numeric-expression)

numeric expression	Any numeric expression.

Example

```
10  x=8
20  b$ = OCT$(x)
30  PRINT x; " decimal is "; a$; " octal"
```

Output: 8 decimal is 10 octal

PADC$ ()

Usage
Pad the right side and left side of a string, to a specified string length, with a specified string character. The input string is centered in the returned string.

Syntax
PADC$(string$, length, padChar$)

string$	The input string to be padded.
length	Length of the output string.
padchar$	Character used as the pad character.

PADC$ returns an input string centered in the output string.

Example
a$ = "abc"

b$ = PADC$(a$, 5,"0")

Result: b$ = "0abc0"

PADL$ ()

Usage
Pad the left side of a string, to a specified string length, with a specified string character.

Syntax
PADL$(string$, length, padChar$)

string$	The input string to be padded.
length	Length of the output string.
padchar$	Character used as the pad character.

PADL$ returns an input string right-justified in the output string.

Example

a$ = "aBc"

b$ = PADL $(a$, 5,"0")

Result: b$ = "00aBc"

b$ = PADL $(a$, 7,"C")

Result: b$ = "CCCCaBc"

b$ = PADL $(a$, 3,"C")

Result: b$ = "aBc"

PADR$ ()

Usage
Pad the right side of a string, to a specified string length, with a specified string character.

Syntax
PADR$ (string$, length, padchar$)

string$ The input string to be padded.

length Length of the output string.

padchar$ Character used as the pad character.

PADR$ returns an input string left-justified in the output string.

Example
a$ = "aBc"

b$ = PADR$(a$, 5,"0")

Result: b$ = "aBc00"

b$ = PADR$(a$, 7,"C")

Result: b$ = "aBcCCCC"

RIGHT$()

Usage
Returns the specified number of rightmost characters in a string. If you specify a number of characters greater than or equal to the string's length, the entire string is returned.

Syntax
RIGHT$(stringexpression$,n%)

stringexpression$ Any string expression.

n% Number of characters to return. The range is 0 to 80.

Example
```
10  a$ = "JAGXTREME BASIC"
20  PRINT RIGHT$(a$, 5)
```

Output: BASIC

RTRIM$ ()

Usage
Removes spaces from the end of the string.

Syntax
RTRIM$ (stringexpression$)

stringexpression$ Any string expression.

Example
10 a$ = "Hello Cambridge "

20 b$ = RTRIM$ (a$).

Result: b$ = "Hello Cambridge"

SPACE$()

Usage
Returns a string of spaces. Used to indent text.

Syntax
SPACE$(n%)

n% The number of spaces you want in the string. The range is 0 to 80.

Example
```
10 FOR i% = 1 TO 5
20 x$ = SPACE$ (i%)
30 PRINT x$; i%
40 NEXT i%
```

STRING$()

Usage
Returns a string of a specified length made up of a repeating character. Used to create underlines, rows of asterisks, etc.

Syntax
STRING$(length%,{ascii-code% I stringexpression$})

length% The length of the string.

ascii-code% The ASCII code of the repeating character.

stringexpression$ The character you want to repeat.

Example
```
10  PRINT STRING$(5, "-");
```
Output: -----

STR$ ()

Usage
Returns a string representation of a number. Used to manipulate a number as a string and to apply string functions to the number for validation and formatting.

Syntax
STR$(numeric-expression)

numeric expression Any numeric expression.

Example
```
10 NUMBER! = 2.5
20 NUM$ = STR$(NUMBER!)
30 PRINT "XXXXX"
40 PRINT NUM$
50 PRINT LEN (NUM$)
```

Output: XXXXX, 2.5, 3

UCASE$ ()

Usage
Converts a string to upper case.

Syntax
UCASE$(stringexpression$)

stringexpression$ Any string expression.

Example
10 A$ = "good morning, sunshine"

20 A$ = ucase$ (a$)

Result: A$="GOOD MORNING, SUNSHINE"

VAL()

Usage
Converts a numeric string to a number. Enables a program to accept numeric input as a string, use various string functions to validate the input, and then convert the input back to a number for use in calculations.

Syntax
VAL(stringexpression$)

stringexpression$ Any numeric string expression.

Example
10 PRINT VAL("76")

Output: 76

Simple I/O Commands

One of the most important parts of your program is its ability to interface with the terminal operator. JagBASIC supports several simple input/output commands. These commands provide an interface between JagBASIC programs and users. These commands enable your program to:

- Sound the terminal beeper on a specified input or output
- Generate prompts
- Accept user input from the keyboard
- Check for key presses

The beeper tone can be used to signify a warning to a user or to provide positive reinforcement. This simple command enables your program to interactively interface with the user through the use of sound.

The INKEY$, INPUT and LINE INPUT commands enable the program to accept keyboard input.

INKEY$—command checks to see if a key has been pressed. Program execution is not interrupted.

INPUT—command pauses the program's execution while the user enters numeric or character data. Data is assigned to one or more variables of the appropriate type. Program execution resumes when the user presses **ENTER**.

Character display on the terminal's lower display is accomplished through the PRINT command.

This section discusses the following simple input/output commands.

Command	Usage
BEEP	Sounds the terminal beeper tone for the specified milliseconds.
INKEYS	Returns a single keystroke from either the keyboard or keypad as a string.
INPUT	Reads input from the keyboard, serial port, or a file.
KEYSRC	Reports the source of the latest keystroke read by the JagBASIC application through an INPUT or INKEY$.
PRINT	Writes data to the lower display or to a sequential file.
PRINT USING	Writes formatted output to the terminal display or to a file.

TIPS

In order for JagBASIC to use the numeric keypad, either the operator must assign the keypad to JagBASIC using the setup menus or the JagBASIC program must assign the keypad to itself by setting an appropriate value in bas10.

The JagBASIC keyboard input statement supports inputting alphabetic characters using the numeric keypad and the **SELECT** key. Before issuing the input statement, the JagBASIC program must disable the control panel using the **SELECT** key by setting bas87 = 0.

A JagBASIC program may read the function keypad using the keyboard input statement. The function keys operate as follows:

FUNCTION (01), **MEMORY** (03), **TARE** (04), and **ZERO** (07) keys— Terminate the input statement. The input statement returns the key value for the terminating key at the end of the input string.

ESCAPE (02) key—Terminates the input. To use the **ESCAPE** key, the JagBASIC program must disable the control panel using the **ESCAPE** key setting bas86 = 0. The input statement appends the **ESCAPE** key value to the end of the input string.

SELECT (05) key—Facilitates the entry of alphabetic characters through the keypad. To use the **SELECT** key, the JagBASIC program must set bas87 = 0. The **SELECT** key selects the alphabetic characters as shown on the keypad overlay. It does not terminate the input. The input statement does not return a key value for the **SELECT** key in the input string.

CLEAR (06) key—Performs a backspace-erase on the input string. It does not terminate the input. The input statement does not place the **CLEAR** key value in the input string.

ENTER (08) key—Terminates the input statement. The input statement does not return the **ENTER** key value in the input string.

To get key input data from the keypad, you could use the following program:

```
10 DEFSHR escape,bas86

20 DEFSHR select,bas87

30 DEFSHR keyboard,bas10

40 escape=0:REM this enables entry of escape key to JagBASIC

50 select=0:REM this enables entry of alphabetic data to JagBASIC

60 keyboard=1:REM this assigns keypad to JagBASIC
```

```
70 INPUT "enter";a$
80 IF a$="" then GOTO 70
90 termchar%=ASC(right$(a$,1))
100 IF termchar% < 8 THEN LPRINT "function key = ";termchar%
110 LPRINT "input string = ";a$
120 GOTO 70
```

JagBASIC has these special function key values for the QWERTY keyboard keys. These special keys terminate the input.

Note: Setting Shared Data trigger s_60b=1 disables the QWERTY positioning keys in the JagBASIC INPUT statement. Positioning key are key values 0x09 to 0x12.

LEFT_ARROW	=	0×09		
RIGHT_ARROW	=	0×0A		
INSERT_KEY	=	0×0B		
HOME_KEY	=	0×0C		
END_KEY	=	0×0D		
DELETE_KEY	=	0×0E		
UP_ARROW	=	0×0F		
DOWN_ARROW	=	0×10		
PAGE_UP	=	0×11		
PAGE_DOWN	=	0×12		
F1_KEY	=	0×13		
F2_KEY	=	0×14		
F3_KEY	=	0×15		
F4_KEY	=	0×16		
F5_KEY	=	ZERO_KEY	=	0×07
F6_KEY	=	FUNCTION_KEY	=	0×01
F7_KEY	=	SELECT_KEY	=	0×05
F8_KEY	=	CLEAR_KEY	=	0×06
F9_KEY	=	TARE_KEY	=	0×04
F10_KEY	=	MEMORY_KEY	=	0×03
F11_KEY	=	0×17		
F12_KEY	=	0×18		

BEEP

Usage
Sounds the terminal beeper tone for the specified milliseconds. Used to signal an error or warn the user of the consequences of an action.

Syntax
BEEP milliseconds

milliseconds The number of milliseconds that you want the tone to sound.

Example

```
10 FOR I% = 1 TO 20
20 BEEP 30
30 SLEEP 100
40 NEXT I%
```

INKEY$

Usage
Reads a character from the keyboard or keypad. This commands enables your program to respond to special keys without interrupting program execution. INKEY$ returns a single keystroke from either the keyboard or keypad as a string. As many as 10 keystrokes can be stored in the buffer. If the keystroke was an ASCII character or an extended character, the string is 1-byte.

If there is no keystroke available in the buffer, INKEY$ returns a null string. If you want to retrieve a key and determine if it has one of several values, you must save the keystroke in a JagBASIC variable, as follows:

```
10  c$=INKEY$
20  IF c$=CHR$(1) THEN PRINT "function key": GOTO 10
30  IF c$=CHR$(2) THEN PRINT "escape key": GOTO 10
40  IF c$="1" THEN PRINT "1 key": GOTO 10
50  IF c$="A" THEN PRINT "A key": GOTO 10
60  IF c$="" THEN PRINT "no keystroke"
70 GOTO 10
```

Syntax
INKEY$

Example 1

```
10 PRINT "Press A to exit..."
20 IF INKEY$ = "A" THEN GOTO 50
30 GOTO 20
50 END
```

Example 2

```
20 A$=INKEY$
30 IF A$="A" THEN GOTO 60
40 IF A$ = "B" THEN GOSUB 1000
50 GOTO 20
60 END
1000 PRINT A$
1010 RETURN
```

INPUT

Usage
Reads data input from the keyboard. The program accepts character input from the keyboard until the user presses a termination character, such as Enter. The prompt can tell the user what type of information to enter. There are several prompting options with the prompt string. The prompt can specify menu selections, default values, and its appearance on the lower display.

Input reads data from the terminal keyboard, the keypad, or both. The JagBASIC keyboard device must be selected through the setup menus.

Syntax

INPUT [;] ["prompt"{; | ,}] variablelist

prompt	An optional literal string that is displayed on the lower terminal display before the user enters data.
variablelist	Comma delimited list of variables to which the input is assigned.
semicolon [;]	Causes the question mark to be displayed at the end of the prompt.
comma [,]	Suppresses the question mark at the end of the prompt.
caret {^}	When used in the prompt, the prompt will be displayed during input and identifies menu selections. Individual selections within a menu selection list may be separated by a comma, colon, semicolon, or space.

Keyboard Input Example #1

```
110 LPRINT "(^) keeps prompt on display during key input,  (;) generates ?"
120 DIM a$(5)
130 a$(3)="^enter"
140 INPUT a$(3);b$
150 LPRINT "input = ";b$
```

Keyboard Input Example #2

```
210 LPRINT "Does not keep prompt on display during key input, (,) supresses ?"
220 c$="hello "
230 INPUT c$,b$
240 LPRINT "input = ";b$
```

Keyboard Input Example #3

```
310 LPRINT "(^) keeps prompt on display during input, (;) generates ?"
320 INPUT "^hello";b$
330 LPRINT "input = ";b$
```

Keyboard Input Example #4

```
410 LPRINT "(,) keeps print message on display only until key input begins"
420 b$="hello"
430 PRINT "enter? ";b$
440 INPUT ,c$
450 LPRINT "input = ";c$
```

Keyboard Input Example #5

```
510 LPRINT "Setup an input default, keep prompt on display"
520 LPRINT "Enter key accepts the default, or key in new data"
530 a$(4)="^type ^ default"
```

```
540 INPUT a$(4),b$
550 LPRINT "input = ";b$
```

Keyboard Input Example #6

```
610 LPRINT "Setup an input default, keep prompt on display"
620 LPRINT "Enter key accepts the default, or key in new data"
630 b$="default"
640 INPUT "^type^"+b$;b$
650 LPRINT "input = ";b$
```

Keyboard Input Example #7

```
710 LPRINT "Select from a list of inputs, keep prompt on display"
720 LPRINT "Enter key accepts the selection"
30 LPRINT "Any other key advances to next selection"
40 LPRINT "Input variable contains the default value"
50 b$="no"
60 INPUT "^type^ yes,no,maybe";b$
70 LPRINT "input = ";b$
```

Keyboard Input Example #8

```
810 LPRINT "Select from a list of inputs, keep prompt on display"
820 LPRINT "Enter key accepts the selection"
830 LPRINT "Any other key advances to next selection"
840 LPRINT "Input variable contains the default value"
850 b%=4
860 a$(5)="^Number^1,2,3,4,5,6,7,8,9,10"
870 INPUT a$(5);b%
880 LPRINT "input = ";b%
```

Keyboard Input Example #9

```
910 LPRINT "Set integer default value with a template"
920 b%=100
930 INPUT "^type^####" ; b%
940 LPRINT "input = ";b%
```

Keyboard Input Example #10

```
1010 LPRINT "Set double float value with a template"
1020 b#=100.55
```

```
1030 INPUT "^type^ ####.###" ; b#
1040 LPRINT "input = ";b#
```

Keyboard Input Example #11

```
1110 LPRINT "Set string value with a template"
1120 a$(1)="happy trails"
1130 INPUT "^enter^ !!!!!!!!!" ; a$(1)
1140 LPRINT "input = ";a$(1)
```

KEYSRC

Usage
Reports the source of the latest keystroke that has been read by the JagBASIC application through an INPUT or INKEY$ command.

Syntax
KEYSRC()

Returns:

0 = None so far
1 = Keypad
2 = QWERTY Keyboard
3 = Serial Keyboard Input

Example

```
10 C$=inkey$
20 IF c$<>"" AND KEYSRC()=1 THEN PRINT "Keypad Input"
30 GOTO 10
```

PRINT, PRINT USING

Usage
PRINT writes data to the lower terminal display, to a sequential file, or outputs data to the specified serial port.

PRINT USING writes formatted output to the terminal display or to a file. A template is defined that specifies the length and format of each item to be displayed.

Syntax
PRINT [#filenumber%,] expressionlist [{;}]

PRINT [#filenumber%,] USING formatstring$; expressionlist [{;}]

PRINT "expression"

PRINT USING "####.##", formatstring$

PRINT [#filenumber%], string$

#filenumber% The number of an open sequential file. If the file number is omitted, PRINT writes to the lower terminal display. If the filenumber is a Com Port, then PRINT command outputs data to the specified serial port.

expressionlist	List of one or more numeric or string expressions to print.
semicolon {;}	Means print immediately after the last value. The absence of a semicolon {;} means to insert a new line.
formatstring$	A string expression containing characters that format a numeric expression.
#	Digit position.
.	Decimal point position.
^	Prints in exponential format.
-	Space.
+	Sign.
	Other characters are printed as literal data in the output.
	Use these characters to format string expressions
!	Prints corresponding characters of string.
\ \	Prints first n characters of string, where n is the number of blanks between the slashes.
expression	Any character or numeric expression.
string$	Any string expression.

Example

```
10  netto=10.0
20  brutto=20.0
30  PRINT USING "netto_#####.## ___brutto_#####.##";netto;brutto
40  a#=123.456789:b#=87.54321:c#=5.555
50  PRINT USING "$###.## __$###.## __$###.##";a#;b#;c#
70  PRINT USING "+###.## __$###.## __+###.##";a#;b#;c#
80  a#= -123.456789
90  PRINT USING "$###.##";a#
100 PRINT USING "+###.##";a#
110 a%=4567:b%=12:c%=1:d%=123
120 PRINT USING "_###";a%
121 PRINT USING "_###";b%
122 PRINT USING "_###";c%
123 PRINT USING "_###";d%
130 PRINT USING "+###.##";a%
140 a%= -4567
150 PRINT USING "#######";a%
151 PRINT USING "######";a%
152 PRINT USING "#####";a%
160 PRINT USING "+###.##";a%
170 a$="abcdefghijklmnopqrstuvwxyz"
```

```
180 PRINT USING "!!!!";a$
190 PRINT USING "\      \ __\            \";a$;a$
200 PRINT USING "_^^^^  ___^^^^^";a#;b#
```

Serial I/O Commands

In order for JagBASIC to access a remote serial port, you must set up the terminal with either a demand print or custom print connection. Use the setup menus at the remote terminal to set these options.

JagBASIC has an enhanced serial I/O capability, including file-type I/O statements and remote terminal support.

JagBASIC can read and write to local serial ports. In addition, JagBASIC program may read and write to serial ports on remote JAGXTREME terminals within a cluster. The terminal routes the serial I/O messages across Ethernet to the remote terminal containing the serial port. Remote serial I/O allows sharing of devices, such as printers or host connections, among all terminals in a cluster.

A JagBASIC application using the remote serial I/O must be prepared to handle "offline" error situations that do not occur in local serial I/O. You cannot use asynchronous events with remote serial. All operations are performed synchronously.

JagBASIC's serial input and output commands enable you to:

- Access files or serial ports.
- Close a file or serial port.
- Flush received data in the BIOS serial input buffer.
- Read input from the serial port.
- Output data to a terminal serial COMx port.
- Print formatted output on the LPRINT device.
- Output data to the specified serial port.
- Specify the width of a printed line.
- Format lines of text by inserting specified amounts of space between values.

This section discusses the following serial input/output commands

Command	Usage
CKSUM$ ()	Generates the checksum of a string and returns the checksum in string format.
CLOSE	Closes a file or serial port.
COMBITS ()	Reads the Modem input status of Com 3
CRC$	Generates the CRC of a string and returns the CRC in string format.
FLUSH	Discards received data in the BIOS serial input buffer.
INPUT	Reads input from the serial port.
LPRINT	Outputs data to a terminal serial LPRINT device.
LPRINT USING	Prints formatted output on the LPRINT device.
OPEN	Accesses a file or serial port.
PRINT	Writes data to the lower terminal display, to a sequential file, or to a serial port.
PRINT USING	Writes formatted output to the terminal display or to a file.
PRINT #	Outputs data to the specified file or serial port.
SPC()	Skips a specified number of spaces in a PRINT or LPRINT statement.
TAB	Advances to the specified position
WIDTH	Assigns an output line width to the LPRINT device, serial port, or a file.
WIDTHIN	Dynamically assigns input length for serial I/O device.

TIPS

JagBASIC serial file I/O commands cannot be used to access a serial port for which there is an input or continuous output connection assigned in CONFIG SERIAL setup.

The LPRINT device is serial port configured as the first demand print port for Scale A.

CKSUM$ ()

This function generates the checksum of a string and returns the checksum in string format. It calculates the checksum by adding the lower 7 bits of each byte in the string and taking the 2's complement. It is used for validating sent and received messages.

Syntax

CKSUM$(string1$, [string2$,][string3$,]start%)

string1$ Input string with a maximum length of 80 characters.

string2$ Optional input string with a maximum length of 80 characters.

string3$ Optional input string with a maximum length of 80 characters.

start% Character in the string where checksum starts.

Example

```
OPEN "com2:xpr null trm13 len40" FOR OUTPUT AS as #1

message$= chr$(2)+"hello world"+chr$(3)

message$= message$+cksum$(message$,1)

PRINT#1,message$;
```

CLOSE

When a program executes a CLEAR, END, or RUN statement or its last statement, JagBASIC closes all open files and serial ports. Each open file must be closed by its own CLOSE command.

Usage
Closes an open file or serial port. Use CLOSE after all input and output operations for a file or device are concluded. CLOSE releases the memory space reserved in the buffer for the open file or serial port.

Syntax

CLOSE #filenumber%

filenumber% The number of an open file.

Example

```
10  OPEN "com4:cr" FOR OUTPUT AS #2
20  PRINT #2, "HELLO"
30  CLOSE #2
```

COMBITS ()

Usage
The COMBITS command allows you to read the status of the four modem input signals on the COM3 serial port. You must first open the COM3 serial port using the OPEN command.

Syntax
COMBITS(filenumber)

filenumber File number used in the OPEN command for the COM3 serial port.

COMBITS returns an integer with the following bit values OR"ed together. The bit value is set to one.

Example

```
OPENn "com2:" FOR OUTPUT AS #1
a%=combits(1)
```

CRC$

Usage
CRC$ computes a 16 bit CRC on the message text and returns a 4-character string that contains the CRC in ASCII format. The CRC is used primarily with serial communications to ensure that a message is transmitted without errors. The CRC calculation is a CCITT method that uses an "exclusive OR" hashing method with a lookup table. The CRC calculation starts with the first byte and proceeds sequentially to the last byte of the message text. CRC$ uses the following procedure to calculate and return CRC:

- "Exclusive OR" the high-order byte of the current CRC with the next byte of the message text.
- Use resulting 8-bit value as an index into the lookup table to get 16-bit table value.
- Shift the low-order byte of current CRC to the high-order byte and "exclusive OR" the result with the 16-bit value from step 2. This becomes the new current CRC.
- Go to step 1 and repeat the calculation for each byte of the message.
- "OR" each 4-bit nibble of the 16-bit CRC with a hex 30 to convert the CRC to four printable ASCII characters. Start with low-order byte then convert high order byte last.

The following table is used for the calculating the CRC.

0x0000,	0x1021,	0x2042,	0x3063,	0x4084,	0x50A5,	0x60C6,	0x70E7,
0x8108,	0x9129,	0xA14A,	0xB16B,	0xC18C,	0xD1AD,	0xE1CE,	0xF1EF,
0x1231,	0x0210,	0x3273,	0x2252,	0x52B5,	0x4294,	0x72F7,	0x62D6,
0x9339,	0x8318,	0xB37B,	0xA35A,	0xD3BD,	0xC39C,	0xF3FF,	0xE3DE,
0x2462,	0x3443,	0x0420,	0x1401,	0x64E6,	0x74C7,	0x44A4,	0x5485,
0xA56A,	0xB54B,	0x8528,	0x9509,	0xE5EE,	0xF5CF,	0xC5AC,	0xD58D,
0x3653,	0x2672,	0x1611,	0x0630,	0x76D7,	0x66F6,	0x5695,	0x46B4,
0xB75B,	0xA77A,	0x9719,	0x8738,	0xF7DF,	0xE7FE,	0xD79D,	0xC7BC,
0x48C4,	0x58E5,	0x6886,	0x78A7,	0x0840,	0x1861,	0x2802,	0x3823,
0xC9CC,	0xD9ED,	0xE98E,	0xF9AF,	0x8948,	0x9969,	0xA90A,	0xB92B,
0x5AF5,	0x4AD4,	0x7AB7,	0x6A96,	0x1A71,	0x0A50,	0x3A33,	0x2A12,
0xBDFD,	0xCBDC,	0xFBBF,	0xEB9E,	0x9B79,	0x8B58,	0xBB3B,	0xAB1A,
0x6CA6,	0x7C87,	0x4CE4,	0x5CC5,	0x2C22,	0x3C03,	0x0C60,	0x1C41,
0xEDAE,	0xFD8F,	0xCDEC,	0xDDCD,	0xAD2A,	0xBD0B,	0x8D68,	0x9D49,
0x7E97,	0x6EB6,	0x5ED5,	0x4EF4,	0x3E13,	0x2E32,	0x1E51,	0x0E70,
0xFF9F,	0xEFBE,	0xDFDD,	0xCFFC,	0xBF1B,	0xAF3A,	0x9F59,	0x8F78,
0x9188,	0x81A9,	0xB1CA,	0xA1EB,	0xD10C,	0xC12D,	0xF14E,	0xE16F,
0x1080,	0x00A1,	0x30C2,	0x20E3,	0x5004,	0x4025,	0x7046,	0x6067,
0x83B9,	0x9398,	0xA3FB,	0xB3DA,	0xC33D,	0xD31C,	0xE37F,	0xF35E,
0x02B1,	0x1290,	0x22F3,	0x32D2,	0x4235,	0x5214,	0x6277,	0x7256,
0xB5EA,	0xA5CB,	0x95A8,	0x8589,	0xF56E,	0xE54F,	0xD52C,	0xC50D,
0x34E2,	0x24C3,	0x14A0,	0x0481,	0x7466,	0x6447,	0x5424,	0x4405,
0xA7DB,	0xB7FA,	0x8799,	0x97B8,	0xE75F,	0xF77E,	0xC71D,	0xD73C,
0x26D3,	0x36F2,	0x0691,	0x16B0,	0x6657,	0x7676,	0x4615,	0x5634,
0xD94C,	0xC96D,	0xF90E,	0xE92F,	0x99C8,	0x89E9,	0xB98A,	0xA9AB,
0x 5844	0x4865,	0x7806,	0x6827,	0x18C0,	0x08E1,	0x3882,	0x28A3,
0xCB7D,	0xDB5C,	0xEB3F,	0xFB1E,	0x8BF9,	0x9BD8,	0xABBB,	0xBB9A,
0x4A75,	0x5A54,	0x6A37,	0x7A16,	0x0AF1,	0x1AD0,	0x2AB3,	0x3A92,
0xFD2E,	0xED0F,	0xDD6C,	0xCD4D,	0xBDAA,	0xAD8B,	0x9DE8,	0x8DC9,
0x7C26,	0x6C07,	0x5C64,	0x4C45,	0x3CA2,	0x2C83,	0x1CE0,	0x0CC1,
0xEF1F,	0xFF3E,	0xCF5D,	0xDF7C,	0xAF9B,	0xBFBA,	0x8FD9,	0x9FF8,
0x6E17,	0x7E36,	0x4E55,	0x5E74,	0x2E93,	0x3EB2,	0x0ED1,	0x1EF0

Syntax

CRC$(string$)

string$ Input string with a maximum length of 160 characters in the JAGXTREME terminal.

Example 1

```
OPEN "com2:xpr null" FOR OUTPUT AS #1

message$= CHR$(2)+"hello world"+CHR$(3)

message$= message$+CRC$(message$)

PRINT #1,message$;
```

Example 2

```
message$= chr$(2)+"hello world"+chr$(3)

x$=CRC$(message$)

message2$="happy trails to you"

y$=CRC$(message2$)

z$=CRC$("a")

LPRINT "x$: ";x$

LPRINT "y$: ";y$

LPRINT "z$: ";z$
```

Output

```
X$: 9==9

Y$: 060?

Z$: 877<
```

FLUSH

Usage
Discards received data in the BIOS serial input buffer.

Syntax
FLUSH #1

Example

```
20 FLUSH #1
```

INPUT

The serial port input can occur asynchronously with the normal program operation. The program execution does not necessarily have to suspend itself while the serial input operation completes using the EVENT option.

Usage
Synchronously reads data bytes from the specified serial communication port into variables until one of the terminating conditions occurs. The terminating conditions are specified in the OPEN statement. Using the EVENT option, the INPUT statement can also be used to asynchronously input data from a serial port.

Syntax
INPUT #filenumber, string variable

#filenumber% Open serial I/O device from which you want to read data.

string variable The input data.

Example 1

```
10 OPEN "com1: tmo5000 len40 trm13 event" FOR INOUT AS #1

30 ON EVENT #1 GOSUB 1000

40 INPUT  #1,a$

.

.  MAIN PROGRAM

.

50 IF INKEY$<>"x" THEN GOTO 50
```

```
60 CLOSE #1
70 END
.
.
1000 LPRINT "serial message";a$
1010 INPUT  #1,a$ :REM start next input
1020 RETURN
```

Example 2

```
10 OPEN "com2: tmo1000 len20 trm13" FOR INPUT AS #1
20 INPUT #1,a$
30 LPRINT "msg="; a$
40 GOTO 20
```

LPRINT

The Configure Serial menu allows you to setup the LPRINT device for JagBASIC. The LPRINT device is the first demand print port for Scale A. When you assign the LPRINT device and the BasTerminal connection to the same serial port, then that serial port operates as an interactive serial port for JagBASIC.

Usage

Outputs the contents of numeric or string variables to a terminal serial port. The first serial port configured as the first demand print connection is used as the output device. The LPRINT statement is most useful for software debugging and for use as an application report printer. When used as an aid in debugging software, error messages are outputted to the LPRINT device.

LPRINT for general serial output is limited by the special handling of three ASCII character codes as listed below:

Name	Hex Code	JagBASIC LPRINT Action
repeat	7F	Used as an escape character in the "tab to column x" feature. The character following the repeat character specifies how many space characters are needed to get to the desired column.
tab	09	Translates into 1 to14 spaces, as required to reach the next "tab stop".
newline	0A	Translates into a <cr><lf> combination.

LPRINT sends output directly to an output device. LPRINT enables you to print strings, numbers, and so on the printer, just as PRINT enables you to display these items on the lower terminal display.

Syntax

LPRINT expressionlist [{;}]

expressionlist List of one or more numeric or string expressions to print. Items must be separated by commas or semicolons.

; When used in a list of expressions, the semicolon deter- mines that the next output is printed immediately after the previous one. When used at the end of the LPRINT statement the semicolon determines that the print head does not move to the next line after printing.

Example
10 LPRINT CHR$(10);"Sample Line Print"

Output: Sample Line Print

LPRINT USING

Usage
Prints formatted output on the LPRINT device and specifies the length and format of each item printed. LPRINT USING creates a template string that filters and formats your output. LPRINT USING functions similarly to PRINT USING. PRINT USING is discussed in the Simple I/O Commands section of this chapter.

Syntax
LPRINT USING formatstring$; expressionlist [{;}]

formatstring$ A string expression containing characters that format a numeric expression.

#	Digit position.
.	Decimal point position.
^	Exponential format.
+	Sign.

Or a string expression

!	Print corresponding characters of string.
\	Print first n characters of string, where n is the number of blanks between the slashes.

expressionlist List of one or more numeric or string expressions to print.

; When used in a list of expressions, the semicolon determines the next output is printed immediately after the previous one. When used at the end of the LPRINT USING statement, the semicolon determines that the print head does not move to the next line after printing.

Example
```
10  netto=10.0
20  brutto=20.0
30  LPRINT USING "netto_#####.## ___brutto_#####.##";netto;brutto
40  a#=123.456789:b#=87.54321:c#=5.555
50  LPRINT USING "$###.## __$###.## __$###.##";a#;b#;c#
70  LPRINT USING "+###.## __$###.## __+###.##";a#;b#;c#
80  a#= -123.456789
90  LPRINT USING "$###.##";a#
100 LPRINT USING "+###.##";a#
110 a%=4567:b%=12:c%=1:d%=123
120 LPRINT USING "_###";a%
121 LPRINT USING "_###";b%
122 LPRINT USING "_###";c%
123 LPRINT USING "_###";d%
130 LPRINT USING "+###.##";a%
```

```
140 a%= -4567
150 LPRINT USING "#######";a%
151 LPRINT USING "######";a%
152 LPRINT USING "#####";a%
160 LPRINT USING "+###.##";a%
170 a$="abcdefghijklmnopqrstuvwxyz"
180 LPRINT USING "!!!!";a$
190 LPRINT USING "\      \ __\            \";a$;a$
200 LPRINT USING "_^^^^  ___^^^^^";a#;b#
```

OPEN

You must set up the serial connections used by the terminal operating system with the Configure Serial in the terminal setup menus. Demand print and custom print ports can be shared by JagBASIC and the terminal operating system. If you attempt to open a serial port that is in the middle of a demand print, for example, you will get a "No Remote Access" error. You must handle this error with an ON ERROR GOTO statement. Once the demand print is complete, you will be able to open the serial port. Similarly, you will not be able to do a demand print while JagBASIC has the port open.

Usage

Prepares a serial port for use as a file device. You can access a serial port that you have set up as a demand print serial connection or a custom print serial connection. You cannot access a serial port from JagBASIC if it has been set up as a continuous output connection or as an input connection. If the serial port is on the local terminal, you can access the serial port even if it is not set up in a connection.

The OPEN command allows you to specify the remote terminal address and the serial port address on the remote terminal. When it issues the OPEN command, the JagBASIC program is establishing exclusive access to the remote serial port as long as it has the serial port open. If another terminal has already opened the serial port, the JagBASIC program will get an error status back indicating there is a file-sharing error. In order to effectively share a serial port among several terminals, you should open the serial port, quickly perform the I/O, and then close the serial port to make it available to another terminal.

Syntax

```
OPEN "com1: tmo5000 len40 trm13 cr event" FOR INPUT AS #1
OPEN "j2/com1: tmo5000 len40 trm13 cr" FOR INPUT AS #1
OPEN "com2: null xpr tmo100" FOR INPUT AS #1
```

com1, com2, com3, and com4 — File names which specify the serial port to be used for communications.

tmo — Specifies the time-out value to wait for a serial input message in decimal milliseconds. The default value is zero milliseconds, or no time-out value. The maximum time-out value is 30,000 milliseconds.

len — Specifies the maximum input length for a serial input message. The maximum length is 80 bytes, which is the maximum string size in JagBASIC. The default length is 80 bytes.

trm — Specifies an optional terminating character for the serial input message. Its value is specified in decimal. When the input command encounters the terminating character, it returns the characters up to and including the terminating character in the serial message as a string variable.

cr — Specifies that a carriage return character is to be inserted at the end of any serial output message.

event — Allocates an event which may trigger an event processing routine when a serial input operation completes.

xpr Selects the "express print" option. Normally, JagBASIC sends PRINT data to a serial port when either it encounters a "new line" character in the print data or the print data length exceeds the WIDTH value. This option causes JagBASIC to send the PRINT data to the serial port immediately at completion of the PRINT statement, even when there is no terminating "new line" character.

null Enables the inputting and outputting of NULL (0) characters through JagBASIC serial I/O. Since the NULL character is a terminator for JagBASIC strings, you must send and receive a special sequence of characters for the NULL character. The sequence "DLE 0xff" represents the NULL character in the JagBASIC application. The sequence "DLE DLE" represents a single DLE character. The following statements transmit a NULL character embedded in each print statement.

```
open "com2:null xpr tmo100" for output as #1
        print #1,chr$(16)+chr$(255)+"hello"
```

The following statements transmit a single DLE character embedded in the print statement.

```
open "com2:null xpr tmo100" for input as #1
print #1, "hello"+chr$(16)+chr$(16)+"dolly"
```

The following statements can receive a single NULL character in the input string.

```
10 OPEN "com2:null xpr tmo1000" FOR INPUT AS #1
20 INPUT #1,a$
30 IF len(a$)=0 THEN LPRINT "timeout":GOTO 20
40 IF len(a$)<>2 THEN GOTO 20
50 IF asc(mid$(a$,1,1))=16 and asc(mid$(a$,2,1))=-1 THEN LPRINT "NULL"
60 GOTO 20
```

input, output The mode can be either input or output. No matter which is chosen, you can do input or output to the specified serial device.

#n The internal device number with a value between 0 and 7, inclusive.

j1, j2, j3, j4, j5, and *J6* Terminal addresses which specify remote terminal containing the remote serial port.

Example 1

```
10 ON ERROR GOTO 1000

20 OPEN "COM1: TMO3000 TRM13" FOR INPUT AS #1

30 OPEN "COM4: CR" FOR OUTPUT AS #2

40 FLUSH #1

50 INPUT #1,A$

60 PRINT #2,A$

70 IF INKEY$<>"C" THEN GOTO 50

80 CLOSE #1

90 CLOSE #2

100 END

1000 SLEEP 500

1010 IF ERR()=32 AND ERL()=20 THEN GOTO 20

1020 IF ERR()=32 AND ERL()=30 THEN GOTO 30
```

```
1030 PRINT "FATAL ERROR"
1040 SLEEP 2000
1050 END
```

Example 2

```
10 OPEN "COM2: TMO5000 TRM13 LEN10 CR" FOR INPUT AS #1
20 FLUSH #1
30 PRINT #1,"SEND SERIAL INPUT"
40 INPUT #1,A$
50 PRINT #1,"SERIAL OUTPUT DATA  ";A$
60 GOTO 40
```

PRINT, PRINT USING

Usage

PRINT writes data to the lower terminal display, to a sequential file, or outputs data to the specified serial port.

PRINT USING writes formatted output to the terminal display or to a file. A template is defined that specifies the length and format of each item to be displayed.

Syntax

PRINT [#filenumber%,] expressionlist [{;}]

PRINT [#filenumber%,] USING formatstring$; expressionlist [{;}]

PRINT expression

PRINT USING "####.##", formatstring$

PRINT [#filenumber%], string$

#filenumber%	The number of an open sequential file. If the file number is omitted, PRINT writes to the lower terminal display. If the filenumber is a Com Port, then PRINT command outputs data to the specified serial port.
expressionlist	List of one or more numeric or string expressions to print.
semicolon {;}	The absence of a semicolon {;} at the end of the line means to insert a new line.
formatstring$	A string expression containing characters that format a numeric expression.
#	Digit position.
.	Decimal point position.
^	Prints in exponential format.
-	Space.
+	Sign.

Other characters are printed as literal data in the output.

Use these characters to format string expressions

!	Prints corresponding characters of string.

\ \	Prints first n characters of string, where n is the number of blanks between the slashes.
expression	Any character or numeric expression.
string$	Any string expression.

Example

```
10  netto=10.0
20  brutto=20.0
30  PRINT USING #1, "netto_#####.## ___brutto_#####.##";netto;brutto
40  a#=123.456789:b#=87.54321:c#=5.555
50  PRINT USING #1,"$###.## __$###.## __$###.##";a#;b#;c#
70  PRINT USING #1, "+###.## __$###.## __+###.##";a#;b#;c#
80  a#= -123.456789
90  PRINT USING #1, "$###.##";a#
100 PRINT USING #1, "+###.##";a#
110 a%=4567:b%=12:c%=1:d%=123
120 PRINT USING #1, "_###";a%
121 PRINT USING #1,"_###";b%
122 PRINT USING #1, "_###";c%
123 PRINT USING #1, "_###";d%
130 PRINT USING #1,"+###.##";a%
140 a%= -4567
150 PRINT USING #1, "#######";a%
151 PRINT USING #1,"######";a%
152 PRINT USING #1,"#####";a%
160 PRINT USING #1,"+###.##";a%
170 a$="abcdefghijklmnopqrstuvwxyz"
180 PRINT USING #1,"!!!!";a$
190 PRINT USING #1,"\     \__\          \";a$;a$
200 PRINT USING #1,"_^^^^  ___^^^^^";a#;b#
210 CLOSE #1
```

PRINT

Usage
Outputs unformatted data to the specified serial port.

Syntax
PRINT comport#1,string$

comport# Number of the serial port.

string$ Any string expression.

Example
```
10 OPEN "COM2: TMO5000 TRM13 LEN10 CR" FOR INPUT AS #1
20 FLUSH #1
30 PRINT #1,"SEND SERIAL INPUT"
40 INPUT #1,A$
50 PRINT #1,"SERIAL OUTPUT DATA  ";A$
60 GOTO 40
```

SPC()

Usage
Displays the specified number of spaces in a PRINT or LPRINT statement. Use SPC to format output for readability.

Syntax
SPC(n%)

n% The number of spaces to display. The range is 1 to 80.

Example
```
10  PRINT "Text1"; SPC(10); "Text2"
```
Output: Text1 Text2

TAB ()

Usage
Advances the cursor to the specified position in a PRINT or LPRINT statement. Use a semicolon (;) to stay on the same line.

Syntax
TAB(n)

n Position to advance to the right.

Example
```
10 LPRINT "COMPANY" TAB(25) "PRODUCT" : PRINT

20 READ A$, B$

30 PRINT A$; TAB(25); B$

40 DATA "METTLER TOLEDO", "JAGBASIC"

RUN
```

Output:

```
COMPANY          PRODUCT
METTLER TOLEDO   JAGBASIC
```

WIDTH

Usage
Assigns an output line width to the LPRINT device, serial port, or a file. Used to limit the line lengths in a file containing a report. Line lengths beyond the established width are wrapped to the next line. The default width is 80 characters.

Syntax
WIDTH [#filenumber%], columns%

#filenumber% The number of an open file. If #filenumber% is not specified, WIDTH applies to the LPRINT device.

columns% The desired width in columns.

Example
```
10 OPEN "COM2:CR" FOR OUTPUT AS #1

20 WIDTH #1, 75
```

WIDTHIN

Usage
Allows you to dynamically reassign the maximum serial input length, as it is defined in OPEN.

Syntax
WIDTHIN #filenumber, length%

#filenumber Open serial I/O device.

length% The desired length. The length can be 0 to 80.

Example

```
10 OPEN "com2: TMO5000 TRM13 LEN10 CR" FOR INPUT AS #1
20 WIDTHIN #1,5
30 INPUT #1, A$
40 LPRINT A$
50 CLOSE #1
```

File Commands

JagBASIC commands perform simple operations such as open and close, as well as complex operations. JagBASIC supports sequential, random, and indexed sequential files. Sequential files are read and written sequentially. Sequential files can have variable length records. You can dynamically change the length of a sequential file by appending records to the end of the file. When you are writing a sequential file, you should frequently close the file so that the file pointers are permanently updated in the RAM disk. Otherwise, you can lose data in the event of a power failure.

Random access files are fixed in length. Records are accessed randomly by number or can be accessed sequentially. Record sizes are fixed in length. You create a random access file by writing it sequentially when you first create the file.

A JagBASIC program can create and access indexed sequential files. Indexed sequential files contain records stored sequentially based on a logical key within a random access file. The records have a fixed length. Indexed sequential files provide keyed access to records within the file. JagBASIC can read, insert, update, or delete records from the file based on the logical key that is stored as part of the record. The JagBASIC interpreter performs a binary search of the records in the file to locate a particular record, providing faster logical access to the records in the file.

This section discusses the following JagBASIC file commands:

Command	Usage
CLOSE	Closes an open file or serial port.
CVI, CVS, CVD	Convert strings to numbers.
DELREC	Deletes a record from the indexed sequential file.
EOF()	Tests for the end of a file.
FIELD	Defines the structure of records to be used in indexed-sequential and random-access file buffers.
GET	Reads a record from the random-access or indexed-sequential file.
INDEXED	Identifies a file as an indexed-sequential file and which field in the record is the index key.
INPUT	Reads input from the keyboard, serial port, or a sequential file.
LINE INPUT#	Reads sequentially all characters of an entire line (up to 80 characters) without delimiters from a sequential file up to the next carriage return into a string variable.
LOC()	Returns the current position within a file.
LOF()	Returns the length of the file.
LSET	Moves data into a random-access file buffer (in preparation for a PUT statement) and left-justifies the value of a string variable.
MKI$, MKS$, MKD$	Convert numbers to numeric strings that can be stored in FIELD statement string variables.
OPEN	Accesses a file.

Command	Usage
PRINT	Writes data to the lower terminal display or to a sequential file.
PRINT USING	Writes formatted output to the terminal display or to a file.
PRINT#	Outputs data to the specified serial port or sequential file.
PUT	Writes a record to the indexed sequential file.
RSET	Moves data into a random-access file buffer (in preparation for a PUT statement) and right-justifies the value of a string variable.
SORTREC	Identifies the file as an indexed sequential file and automatically sorts the records.
WRITE#	Writes data to the LPRINT device or to a sequential file.

TIPS

To perform quick file look-ups based on a logical key, use indexed sequential files.

CLOSE

Each open file must have its own CLOSE command.

When you are writing a indexed-sequential or sequential file, you should frequently close the file to avoid losing data in the event of a power failure.

Usage
Closes an open file or serial port. Only one CLOSE command is permitted per program line.

Syntax
CLOSE #filenumber%

#filenumber% The number of an open file.

Example

```
10  OPEN "LOG" FOR OUTPUT AS #1
20  WRITE #1, "This is saved to the file."
30  CLOSE #1
40  OPEN "LOG" FOR INPUT AS #1
50  INPUT #1, a$
60  PRINT "Read from file: "; a$
70  CLOSE #1
```

CVI, CVS, CVD

Usage
Convert string variable types, created by either the MKD$, MKI$, or MKS$ commands, to numeric variable types. These commands are used after reading the string representation of a double-precision number in a random-access file that contains records defined by the FIELD statement. Because you cannot store numeric values in random-access files, you must convert numbers to strings before storing them and convert them back to numbers when you read the file.

Command	Returns
CVI	Integer
CVS	Single-precision number
CVD	Double-precision number

Syntax
CVI(2-byte-numeric-string)

CVS(4-byte-numeric-string)

CVD(8-byte-numeric-string)

2-byte-numeric string 2-byte string variable created by the MKI$ command

4-byte numeric string 4-byte string variable created by the MKS$ command

8-byte-numeric string 8-byte string variable created by the MKD$ command

Example
```
70 FIELD #4, 4 AS N$, 12 AS B$

80 GET #1
```

DELREC

Usage
Deletes a record from the indexed sequential file. The JagBASIC program must set the logical index into the key field of FIELD variables. DELREC searches the file for a record containing the logical key. If it finds the record, DELREC deletes the record in the FIELD variables. Otherwise, DELREC generates a "RECORD NOT FOUND" error.

Syntax
DELREC #file number

#file number The number of the indexed sequential file.

Example
```
6000 LPRINT "delete some records"

6001 ON ERROR GOSUB 6200

6010 OPEN "testfile" FOR RANDOM AS #1 len=26

6020 field #1,16 AS a$,8 AS b$, 2 AS c$

6030 INDEXED #1,a$

6050 LSET a$=STRING$(16,"A")

6080 DELREC #1
```

```
6090 LSET a$=STRING$(16,"Z")
6120 DELREC #1
6130 END
6200 IF ERR()<>6 THEN END
6210 LPRINT "error line ";ERL()
6220 RETURN
```

EOF()

Usage
Tests for the end of a file. Returns true (nonzero) if the end of a file has been reached. Used to decide whether to continue processing a file.

Syntax
EOF(filenumber%)

filenumber% Number of the file to test.

Example

```
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
20 FOR i% = 1 TO 10
30 WRITE #1, i%, 2 * i%, 5 * i%
40 NEXT i%
50 CLOSE #1
60 OPEN "TEST.DAT" FOR INPUT AS #1
70 WHILE EOF(1) = 0
80 LINE INPUT #1, a$
90 PRINT a$
100 WEND
```

FIELD

The maximum record length for a random access or indexed-sequential file is 200 characters.

Usage
Defines the structure of records to be used in indexed-sequential and random-access file buffers. Records contain various fields. Each field is a location in a record that can be accessed by a field name.

Syntax
FIELD #filenumber%, fieldwidth% AS stringvariable$ [,fieldwidth% AS stringvariable$]

#filenumber% The number of an open file.

fieldwidth% The number of characters in field.

stringvariable$ A variable that identifies the field and contains field data.

Example

```
40 OPEN "FILE" FOR RANDOM AS #1 LEN = 80
50 FIELD #1, 30 AS Name$, 50 AS address$
```

GET

Usage
Reads a record from a random access file by record number into fields defined by the field statement.

Reads a record from the indexed sequential file into the fields defined by a FIELD statement. The program must first set a logical index into the key field of the FIELD variables. GET executes a binary search of the file for a record containing the logical key. If it finds the record, GET returns the record in the FIELD variables. Otherwise, GET generates a "RECORD NOT FOUND" error. You must use an ON ERROR statement to handle these errors.

Syntax
GET #file number[,record number]

#file number Number of the open or sequential file.

record number For random access files, the number of the record to read. If record number is not specified, GET returns the next sequential record.

For indexed-sequential files, the number is typically not specified. When it is not specified, GET returns the record specified in the keyword field of the FIELD statement. The INDEXED or SORTREC command specifies which field is the keyword field. When the record number is specified, the GET statement returns the specified record number. The record number can be variable or a constant.

Example 1
Reading a Random Access File Sequentially

```
10 OPEN "M" FOR RANDOM AS #1 LEN=21
20 FIELD#1, 5 AS ID$, 16 AS MATNAME$
30 WHILE EOF (1)=0
40 GET #1
50 PRINT ID$; TAB (10); MATNAME$;
60 WEND
70 CLOSE #1
80 END
```

Example 2
Indexed Sequential File

```
8000 LPRINT "get some records"
8001 ON ERROR GOSUB 8200
8010 OPEN "testfile" FOR RANDOM AS #1 len=26
8020 field #1,16 as a$,8 as b$, 2 as c$
8030 INDEXED #1,a$
8040 LSET a$=STRING$(15,"A")+"1"
8050 GET #1
8060 LPRINT b$
8070 LSET a$=STRING$(15,"J")+"1"
8080 GET #1
```

```
8090 LPRINT b$
8100 END
8200 IF ERR()<>6 THEN END
8210 LPRINT "error line ";ERL()
8220 RETURN
```

Example 3
Reading Indexed Sequential File sequentially

```
1000 OPEN "testfile" FOR RANDOM AS #2 len=27
1010 FIELD #2, 5 AS a$, 10 AS b$, 12 AS c$
1020 INDEXED #2, b$
1030 r%=0
1040 WHILE NOT EOF(2)
1050 r%=r%+1
1060 GET #2, r%
1070 LPRINT c$
1080 WEND
```

INDEXED

Usage
Identifies which field in the record is the index key. JagBASIC must first OPEN the file as a random access file and define the record format using the FIELD command. The INDEXED command identifies the file as an indexed sequential file.

Syntax
INDEXED #file number,variable name

#file number	The opened random access file.
variable name	Name of the FIELD variable that is the index key.

Example

```
1000 LPRINT "create indexed file"
1010 OPEN "testfile" FOR RANDOM AS #1 len=26
1020 FIELD #1,16 AS a$,8 AS b$, 2 AS c$
1030 INDEXED #1,a$
1040 FOR i% = 10 to 1 step -1
1050 LSET a$=STRING$(16,chr$(64+i%))
1055 LSET b$="00000000"
1060 LSET c$=CHR$(13)+CHR$(10): REM LF/CR
1070 PUT #1
1080 NEXT i%
1090 CLOSE #1
2000 LPRINT "print file"
```

```
2010 OPEN "testfile" FOR INPUT AS #1
2020 WHILE NOT EOF(1)
2030 LINE INPUT #1,x$
2040 LPRINT x$
2050 WEND
```

INPUT

Usage
Reads input from the keyboard, serial port, or a sequential file. When reading a sequential file, the file must be "comma-delimited". That is, commas between items and quotation marks around strings in the file are required.

Syntax
INPUT #filenumber%, variablelist

#filenumber%	Open sequential file from which you want to read data. When no filename is specified, INPUT reads data from the keyboard.
variablelist	List of variables to which input is assigned.

Example

```
100 OPEN "LOG" for output as #1
200 WRITE #1, "Write this to the file."
300 CLOSE #1
400 OPEN "LOG" for input as #1
500 INPUT #1, a$
600 PRINT "Read from file:"; a$
700 CLOSE #1
```

LINE INPUT

Usage
Reads sequentially all characters of an entire line (up to 160 characters) without delimiters from a sequential file up to the next carriage return into string variable.

Syntax
LINE INPUT #filenumber%,string$

#filenumber%	File.
stringvariable	String variable.
string$	String expression.

Example

```
10 OPEN "log" for input a$ #1
20 WHILE eof(1)=0
30 LINE INPUT #1, a$
40 WEND
```

LOC()

Usage
Returns the current pointer position within a file that shows where the next read or write operation will take place.

For random access files, LOC returns the next record number after the last record read from or written to the file.

For sequential input or output, LOC returns the current byte position.

Syntax
LOC(filenumber%) #number

filenumber% The number of an open file.

#number The number of records.

Example
```
200 IF LOC(1)=50 THEN STOP
```

LOF

Usage
Returns the length of a file.

Syntax
LOF (filenumber%)

Filenumber The number of an open file.

Example
```
100 OPEN "TEST" FOR INPUT AS #1
200 size# = LOF(1)
```

LSET

Usage
Moves the value of an expression or variable into a field in a random-access file buffer in preparation for a PUT statement. LSET left-justifies the value of a string variable in the field.

Syntax
LSET stringvariable$ = stringexpression$

stringvariable$ Any string variable or a random-access file field defined in a FIELD statement.

*stringexpression$* The left-justified version of string variable$.

Example
```
1  OPEN "F" FOR RANDOM AS #1 LEN = 10
2  FIELD #1, 5 AS Ls1$, 5 AS Rs1$
3  LSET Ls1$ = "LSET"
4  RSET Rs1$ = "RSET"
5  PUT #1, 1
6  CLOSE #1
```

MKI$, MKS$, MKD$

Usage
Convert numbers to numeric strings that can be stored in FIELD statement string variables. You cannot store numeric values in random-access files. You must convert numbers to strings before storing them. These commands complement the CVI, CVD, and CVS commands which convert the strings back to numbers when you read the file.

Function	Returns
MKI$	2-byte string
MKS$	4-byte string
MKD$	8-byte string

Syntax
MKI$(integer-expression%)

MKS$(single-precision-expression!)

MKD$(double-precision-expression#)

integer-expression%	Any integer number in the range of -32768 to 32767.
single-precision-expression!	Single-precision number in the range of 3.4E-38 to 3.4E+38.
double-precision-expression#	Double-precision number in the range of 7E-308 to 7E+308.

OPEN

Usage
Accesses a file. Files can be sequential, random, or indexed-sequential files stored on the terminal RAMDISK.

Syntax
OPEN file$ [FOR mode] AS #filenumber% [LEN=reclen%]
OPEN file$ [FOR mode] AS #filenumber% [LEN=reclen%]

file$	The name of the file on the RAMDISK.
mode	INPUT, OUTPUT, APPEND, or RANDOM.

Sequential files are opened as INPUT, OUTPUT, or APPEND. Opening a sequential file for OUTPUT creates a new file. Opening a sequential file for APPEND adds new records to the end of an existing file. Random access and indexed sequential files must be opened as RANDOM.

filenumber%	A number in the range 0 through 7 that identifies the file while it Is open.
reclen%	For random access files and indexed-sequential files, this is the record length.

Example

```
100 OPEN "LOG" FOR OUTPUT AS #1
200 WRITE #1, "write this to the file."
300 CLOSE #1
400 OPEN "LOG" FOR INPUT AS #1
500 INPUT #1, a$
600 PRINT "Read from file: "; a$
700 CLOSE #1
```

PRINT, PRINT USING

Usage

PRINT writes data to the lower terminal display, to a sequential file, or outputs data to the specified serial port.

PRINT USING writes formatted output to the terminal display or to a file. A template is defined that specifies the length and format of each item to be displayed.

Syntax

```
PRINT [#filenumber%,] expressionlist [{;}]
PRINT [#filenumber%,] USING formatstring$; expressionlist [{;}]
PRINT "expression"
PRINT USING "####.##", formatstring$
PRINT [#filenumber%], string$
```

#filenumber%	The number of an open sequential file. If the file number is omitted, PRINT writes to the lower terminal display. If the filenumber is a Com Port, then PRINT command outputs data to the specified serial port.
expressionlist	List of one or more numeric or string expressions to print.
;	The absence of a semicolon {;} at the end of a line means to insert a new line.
formatstring$	A string expression containing characters that format a numeric expression.
#	Digit position.
.	Decimal point position.
^	Prints in exponential format.
-	Space.
+	Sign.

Other characters are printed as literal data in the output.

Use these characters to format string expressions

!	Print corresponding characters of string.
\ \	Print first n characters of string, where n is the number of blanks between the slashes.
expression	Any character or numeric expression.
string$	Any string expression.

Example

```
10  netto=10.0
20  brutto=20.0
30  PRINT #1, USING"netto_#####.## ___brutto_#####.##";netto;brutto
40  a#=123.456789:b#=87.54321:c#=5.555
50  PRINT #1, USING"$###.## __$###.## __$###.##";a#;b#;c#
70  PRINT #1, USING"+###.## __$###.## __+###.##";a#;b#;c#
80  a#= -123.456789
90  PRINT #1, USING "$###.##";a#
100 PRINT #1, USING "+###.##";a#
110 a%=4567:b%=12:c%=1:d%=123
120 PRINT USING #1, "_###";a%
121 PRINT USING #1, "_###";b%
122 PRINT USING #1, "_###";c%
123 PRINT USING #1, "_###";d%
130 PRINT USING #1, "+###.##";a%
140 a%= -4567
150 PRINT USING #1, "#######";a%
151 PRINT USING #1, "######";a%
152 PRINT USING #1, "#####";a%
160 PRINT USING #1, "+###.##";a%
170 a$="abcdefghijklmnopqrstuvwxyz"
180 LPRINT USING #1, "!!!!";a$
190 PRINT, USING #1"\      \ __\            \";a$;a$
200 PRINT, USING #1"_^^^^  ___^^^^^";a#;b#
210 CLOSE #1
```

PRINT

Usage
Outputs data to the specified serial port or sequential file.

Syntax
PRINT#1,string$

#1	Serial port or file number.
string$	String expression.

Example

```
10 OPEN "LOG" FOR APPEND a$ #1
20 PRINT #1, "hello"
30 CLOSE #1
```

PUT

Field variables are cleared after the PUT statement.

Usage
Writes records to a random access file.

Writes a record to the indexed sequential file. The JagBASIC program must first set values into the FIELD variables, including the logical key variable. PUT searches the file for a record containing the logical key. If it finds the record, PUT overwrites the existing record with the new data. If there is no existing record with the same key, PUT inserts a new record into file in its proper sequential position.

Field variables are cleared after the PUT statement is run.

Syntax
PUT #file number[,record number]

#file number	Number of the open, random, or indexed sequential file.
record number	Number of the record to write. When a record number is not specified for random access files, JagBASIC writes to the record specified by the indexed field of the field variables. Record number is not used for indexed-sequential files.

Example 1
Random File

```
10 OPEN "IDFILE" FOR RANDOM AS #1 LEN = 19
15 REM added line feed, carriage return for
16 REM printing out file with standard editors,
20 FIELD #1,9 AS FID$, 8 AS FWEIGHT$, 2 AS LFCR$
30 FOR X% = 1 TO 10
35  REM re-initialize record image before each "PUT"
40  LSET FID$ = "000000000" : LSET FWEIGHT$ = "00000000"
50  LSET LFCR$=chr$(13)+chr$(10)
60  PUT #1, X%
70  NEXT X%
80 CLOSE #1
```

```
230 USEREC%=0
240 FOR REC% = 1 TO 10
250  GET #1, REC%
270  IF FID$ = "000000000" THEN USEREC% = REC% : REC%=10
280  IF EOF(1) = 1 THEN REC% = 10
290 NEXT REC%
300 LSET FWEIGHT$ = "12345.6"
310 LSET FID$="JOE TRUCK"
320 LSET LFCR$=chr$(13)+chr$(10)
330 IF USEREC%<>0 THEN PUT #1, USEREC%
340 CLOSE #1
```

Example 2
Indexed-Sequential File

```
3000 LPRINT "write some records"
3010 OPEN "testfile" FOR RANDOM AS #1 len=26
3020 FIELD #1,16 as a$,8 as b$, 2 as c$
3030 INDEXED #1,a$
3050 LSET a$=STRING$(16,"Z")
3060 LSET b$="11111111"
3070 LSET c$=CHR$(13)+CHR$(10)
3080 PUT #1
3090 LSET a$=STRING$(16,"Y")
3100 LSET b$="11111111"
3110 LSET c$=CHR$(13)+CHR$(10)
3120 PUT #1
3170 CLOSE #1
```

RSET

Usage
Moves the value of an expression or variable into a specified field in a random-access file buffer in preparation for a PUT statement. RSET also right-justifies the value of a string variable in the field variable.

Syntax
RSET stringvariable$ = stringexpression$

stringvariable$ Any string variable or a random-access file field defined in a FIELD statement.

*stringexpression$* The right-justified version of string variable$.

Example

```
10  OPEN "F" FOR RANDOM AS #1 LEN = 10
20  FIELD #1, 5 AS Ls1$, 5 AS Rs1$
30  LSET Ls1$ = "LSET"
40  RSET Rs1$ = "RSET"
50  PUT #1, 1
60  CLOSE #1
```

SORTREC

Usage

Identifies the file as an indexed sequential file. Identifies which field is the index field. Sorts the file records in sequential order by key if necessary.

Syntax

SORTREC #file number,variable name

file number Opened random access file.

variable name The FIELD variable used as the index key.

Example

```
1000 LPRINT "create indexed file"
1010 OPEN "testfile" FOR RANDOM AS #1 len=26
1020 FIELD #1,16 as a$,8 as b$, 2 as c$
1040 FOR i% = 10 to 1 step -1
1050 LSET a$=STRING$(16,chr$(64+i%))
1055 LSET b$="00000000"
1060 LSET c$=CHR$(13)+CHR$(10): REM LF/CR
1070 PUT #1
1080 NEXT i%
1100 SORTREC #1,a$
1110 LSET a$=STRING$(16,"J")
1120 GET #1
1130 LPRINT b$
1140 CLOSE #1
2000 LPRINT "print file"
2010 OPEN "testfile" FOR INPUT AS #1
2020 WHILE NOT EOF(1)
2030 LINE INPUT #1,x$
2040 LPRINT x$
2050 WEND
2060 CLOSE #1
```

Output: SORTREC sorted the records into sequential order to make the file an indexed sequential file.

WRITE

Usage
Outputs delimited data to the sequential file. WRITE inserts commas between items and quotation marks around strings as they are written. WRITE writes values in a form that can be read into separate variables by the INPUT statement.

Syntax
WRITE [#filenumber%,] expressionlist

filenumber%	The number of an open sequential file. If the file number is omitted, WRITE writes to the LPRINT device.
expressionlist	One or more variables or expressions.

Example

```
5 ON ERROR GOSUB 80
10 OPEN "log" FOR APPEND AS #1
20 WRITE #1,"write this to log"; "write some more"
30 CLOSE #1
40 OPEN "log" FOR INPUT AS #1
45 WHILE EOF(1) = 0
50 INPUT #1,a$,b$
60 LPRINT "read from log: ";a$,b$
65 WEND
70 CLOSE #1
75 END
80 a$ = "done"
90 RETURN
```

Real-time Process Control Commands

A JagBASIC program can implement "event-driven" processing. A program can execute a particular command or subroutine based on the occurrence of a specified event. A JagBASIC program can also build ladder logic rungs. The terminal's O/S can then use its ladder logic processor to rapidly evaluate the discrete inputs, the discrete outputs, and the associated shared data triggers. The maximum number of rung elements that may be active is 70.

JagBASIC's real-time process control commands enable you to:

You cannot define an event associated with a remote shared data field.

- Allocate and de-allocate events.
- Allocate a keyboard event or timer event.
- Suspend program execution until an event trigger causes program execution to resume.
- Clear outstanding event triggers.
- Disable asynchronous event triggers.

- Re-enable asynchronous event triggers after a critical section of code.
- Return the state of the event.
- Add a rung to the ladder.
- Clear the ladder.

The JagBASIC program allocates events with event names. A maximum of 16 events may be active at any one time. The event name is one of the following:

- a shared data variable name
- the keyword, KEY
- the file number of an open serial communications file, COM1, COM2, COM3, or COM4
- the keyword, TIME

The JagBASIC program can synchronously monitor an event state or wait for the "triggering" of any event in the main line of the program. Changes to local shared data elements, keystrokes, or serial port inputs can trigger events.

Level-sensitive and edge-sensitive discrete shared data fields can trigger events.

Level-sensitive state bit fields trigger events when the terminal O/S writes either 0 or 1 to the field. Applications can use events to monitor when these fields change values.

Edge-sensitive bit fields only trigger events when a 1 is written to the field. The terminal O/S, a PLC host, or a PC host can write these bit fields. Applications can set these discrete shared data bits to issue commands to the terminal O/S. Once the terminal O/S has processed the command, it sets the discrete bit to 0 to rearm the bit for another command. Applications do not typically use events to monitor the state of these bits.

The JagBASIC program can "trap" events asynchronously by designating a specific routine to be executed when the event occurs. The event trapping routines must be short routines that execute quickly then return execution control to the main line of executable code. When you CHAIN from one program to another, the JagBASIC Interpreter automatically clears all events.

This section discusses the following JagBASIC event commands:

Command	Usage
CLREVENT	Clears outstanding event triggers.
DEFSHR EVENT	Allocates a shared data event.
DELEVENT	De-allocates an event.
DISABLE	Disables asynchronous event triggers.
ENABLE	Re-enables asynchronous event triggers after a critical section of code.
EVENT	Allocates a keyboard event or timer event.
EVENTON	Returns the state of the event.
INPUT	Used in conjunction with event commands to implement asynchronous serial input.
NEWLADDER	Clears ladder used by ladder logic processor in JAG UAR terminal O/S
RUNGAND	Adds a rung which represents the AND value of two inputs.
RUNGANDNT	Adds a rung which represents the inverse of the AND value at the inputs.

Command	Usage
RUNGMOV	Adds a rung to the ladder which moves the value of SharedData1 to SharedData2.
RUNGMVNOT	Adds a rung to the ladder which moves the "NOT" value of SharedData1 to SharedData2.
RUNGOR	Adds a rung which represents the OR value of two inputs.
RUNGORNOT	Adds a rung which represents the inverse of the OR value of two inputs.
ON EVENT GOSUB	Enables you to asynchronously monitor an event.
STARTIME	Starts the timer, which specifies the length of the timer in milliseconds.
STOPTIME	Stops a running timer.
WAITEVENT	Suspends program execution until an event trigger causes program execution to resume.

TIPS

An application can monitor discrete edge-sensitive fields to start processing when the scale has read a new weight from the scale base. Trigger t_688 is for Scale A and t_689 is for Scale B. Once it has processed the event, the scale application must set the field back to zero in order to re-enable the trigger for the next event.

Physical discrete input fields are level-sensitive shared data fields that reflect the state of the physical outputs from the terminal. JagBASIC applications can use events to monitor the changing state of the physical inputs.

Physical discrete output fields are level-sensitive shared data fields that reflect the state of the physical outputs from the terminal. JagBASIC applications interface to shared data to set the discrete outputs and would not typically use events to monitor the state of physical discrete outputs.

An application can monitor the rising or falling edge of physical discrete inputs. An event may be processed on either the rising edge when a physical discrete input transitions from a 0 to 1 state, or on the falling edge when the physical discrete input transitions from a 1 to 0 state.

The terminal ladder logic processor continually monitors the state of the physical inputs. It samples the physical discrete inputs once every 55 milliseconds. The ladder logic processor sets the rising or falling edge trigger when it sees a state transition in the discrete input.

The following discrete edge-sensitive triggers can alert an event on either the rising edge or falling edge of a discrete input. Once the application processes the event, it must reset the shared data trigger to 0 to re-enable the next occurrence of the trigger.

DiscreteInputRisingEdge_1 /p_6e0

DiscreteInputRisingEdge_2 /p_6e1

.

.

.

DiscreteInputRisingEdge_12 /p_6ef

```
DiscreteInputFallingEdge_1    /p_6f0
DiscreteInputFallingEdge_2    /p_6f1
.
.
.
DiscreteInputFallingEdge_12   /p_6ff
```

The following sample program uses events to monitor the rising edge and falling edge of discrete input 1. Note that the program resets the triggers to 0 so that they will trigger again.

```
10 DEFSHR event re_1,p_6e0
20 DEFSHR event fe_1,p_6f0
30 re_1=0
40 fe_1=0
50 ON EVENT re_1 GOSUB 1000
60 ON EVENT fe_1 SOSUB 2000
70 IF INKEY$="" then GOTO 70
80 END
1000 TPRINT "rising edge"
1010 re_1=0
1020 RETURN
2000 TPRINT "falling edge"
2010 fe_1=0
2020 RETURN
```

CLREVENT

Usage
Clears outstanding event triggers. The JagBASIC interpreter automatically clears an event trigger upon completion of an event trapping routine for that trigger.

Syntax
CLREVENT [event name]

event name — Name of the specific event that you want to clear. If no event name is specified, all event triggers are cleared.

Example

```
10  CLREVENT  SPFEED%
20  CLREVENT TIME
30  CLREVENT KEY
100 CLREVENT
```

DEFSHR EVENT

Usage
Allocates an event associated with a shared data field. Writing a value to the shared data field triggers a JagBASIC event.

Syntax
DEFSHR EVENT variable name, shared data field name

variable name The variable name. You cannot define an event for a remote shared data field.

shared data field name Local shared data field name.

Example
10 DEFSHR EVENT SPFEED%,s_210

DELEVENT

Usage
De-allocates an event.

Syntax
DELEVENT event name

event name Name of the specific event you want to delete. If no name is specified, all events are deleted.

Example
500 DELEVENT SPFEED%

510 DELEVENT KEY

530 DELEVENT #1

600 DELEVENT

DISABLE

Usage
Disables asynchronous event triggers. This command is used to protect critical sections of code.

Syntax
DISABLE

Example
30 DISABLE

ENABLE

Usage
Re-enables asynchronous event triggers after a critical section of code.

Syntax
ENABLE

Example
50 ENABLE

EVENT

Usage
Allocates a keyboard event or timer event. An event occurs asynchronously from the normal execution of the JagBASIC program.

The keyboard event triggers an event when there is a key available. Use the INKEY$ function to read the key.

The timer event triggers at the expiration of the timer. Use the STARTIME command to start the timer.

Syntax
EVENT [KEY] | [TIME]

key	Keyboard event.
Time	Timer event.

Example 1
```
10 EVENT key
20 WAITEVENT
30 CLREVENT
40 c$=INKEY$
50 WHILE c$<>""
60 TPRINT c$;
70 c$=INKEY$
80 WEND
90 GOTO 20
```

Example 2
```
10 event time
20 ON EVENT time GOSUB 200
30 startime 1000
.
.
200 PRINT "timer expired"
210 RETURN
```

EVENTON

Usage
Returns the state of the event. A zero value indicates the event is in a "non-triggered" state. A nonzero value is the "triggered" state. You must put quotation marks around the event name.

Syntax
EVENTON("event name")

event name Name of the event.

Example
```
100  IF EVENTON("SPFEED%") THEN PRINT "setpoint event"

110 CLREVENT SPFEED%

120 GOTO 100
```

INPUT

Usage
Used in conjunction with the event commands to implement asynchronous serial input, the INPUT command initiates an input operation from a serial port. This can occur asynchronously with the normal application operation. Program execution does not have to be suspended while the serial input operation completes. Upon completion of the serial input, an event trigger alerts the application that the input is complete. The application defines the serial input termination conditions on the OPEN statement (a time-out, reaching a specified input length, or encountering the terminating character in the input stream). The application can use either synchronous or asynchronous event processing routines to complete serial input processing.

After receiving an INPUT message and transferring control to the Event Service Routine, the JagBASIC program must re-prime the input by issuing another INPUT command.

Syntax
INPUT #filenumber, string variable

#filenumber Open sequential file or serial port from which you want to read data. When no filename is specified, INPUT reads data from the terminal keyboard, the terminal keypad, or both. The JagBASIC keyboard device must be selected through the terminal operator setup menus. Commas between items and quotation marks around strings in the file are required.

string variable The input data.

Example
```
10 OPEN "com1:tmo5000 len40 trm13 event" for input as #1

30 ON EVENT #1 GOSBU 1000

40 INPUT #1,a$

.

. MAIN PROGRAM

.

50 IF inkey$ <>"x" then GOTO 50
```

```
60 CLOSE #1
70 END
.
.
1000 LPRINT "serial message";a$
1010 INPUT #1, a$ : REM start next input
1020 RETURN
```

NEWLADDER

Usage
Clears the ladder that is used by the ladder logic processor in the terminal Operating System.

Syntax
NEWLADDER

Example

```
210 REM Ladder based on setpoint
220 NEWLADDER
230 REM Setpoint1 to Out2
240 RUNGMOV 5_210, p_501
```

ON EVENT GOSUB

Usage
Enables you to asynchronously monitor an event and define the Event Service Routine. Upon the occurrence of an asynchronous event, the program execution branches to an event trapping subroutine.

Event trapping routines must be short routines that execute quickly and then return execution control to the main line of program code. The execution of an event trapping subroutine completes without interruption by another asynchronous event. The event trapping routines can occur between any two lines in the main program. Be careful of the variables used in these routines. Temporary variables, such as loop counters, should be unique to the event-trapping routine. Upon exit of the event-trapping routine, the JagBASIC interpreter automatically clears the event that triggered the execution of the routine.

Syntax
ON EVENT event name GOSUB line number

Example 1
Monitoring One Setpoint

```
10 DEFSHR EVENT SPFEED%,s_210
20 ON EVENT SPFEED% GOSUB 1000
.
1000 IF SPFEED%=0 THEN PRINT "SETPOINT REACHED"
1010 RETURN
```

Example 2
Monitoring Multiple Setpoints

```
5 REM Turn discrete outputs on or off  as setpoint coincidence values change.
10 DIM SPFEED%(4)
20 DEFSHR EVENT SPFEED%(1),s_210
30 DEFSHR EVENT SPFEED%(2),s_214
40 DEFSHR EVENT SPFEED%(3),s_218
50 DEFSHR EVENT SPFEED%(4),s_21c
60 DIM DOUT(4)
70 DEFSHR DOUT(1),p_500
80 DEFSHR DOUT(2),p_501
90 DEFSHR DOUT(3),p_502
100 DEFSHR DOUT(4),p_503
110 FOR i%= 1 to 4
120 ON EVENT SPFEED%(i%) GOSUB 1000
130 NEXT i%
.
.MAIN PROGRAM
.
1000 CLREVENT
1010 FOR j%=1 to 4
1020 IF SPFEED%(j%)=0 THEN DOUT(j%)= 0 ELSE DOUT(j%)= 1
1030 NEXT j%
1040 RETURN
```

RUNGAND

Usage
RUNGAND adds a ladder rung to the terminal's Ladder Logic. The ladder is run every 55 milliseconds in the terminal's O/S whenever there is a change in the ladder inputs. The rung inputs are physical discrete inputs or global discrete data from Shared Data. The outputs are physical discrete outputs or global discrete data in Shared Data. This rung takes two inputs, AND's them together, and outputs the value.

Syntax
RUNGAND input1,input2,output

Example
A physical discrete input with "Setpoint1 feeding" to generate a physical discrete output.

```
RUNGAND  p_101,s_210,p_501
```

RUNGANDNT

Usage
RUNGANDNT adds a ladder rung to the Ladder Logic. The ladder is run every 55 milliseconds in the terminal's O/S whenever there is a change in the ladder inputs. The rung inputs are physical discrete inputs or global discrete data from Shared Data. The outputs are physical discrete outputs or global discrete data in Shared Data. This rung takes two inputs, AND's them together, and outputs the inverse value.

Syntax
RUNGANDNT input1,input2,output

Example
Take two physical inputs and generate a physical discrete output.

```
RUNGANDNT p_101,p_102,p_501
```

RUNGMOV

Usage
RUNGMOV adds a ladder rung to the Ladder Logic. The ladder is run every 55 milliseconds in the O/S whenever there is a change in the ladder inputs. The rung inputs are physical discrete inputs or global discrete data from Shared Data. The outputs are physical discrete outputs or global discrete data in Shared Data. This rung takes an input and generates an output with the same value.

Syntax
RUNGMOV input,output

Example
Take a tare on Scale B when a physical discrete input is turned on.

```
RUNGMOV p_103,t_6a0
```

RUNGMVNOT

Usage
RUNGMVNOT adds a ladder rung to the Ladder Logic. The ladder is run every 55 milliseconds in the O/S whenever there is a change in the ladder inputs. The rung inputs are physical discrete inputs or global discrete data from Shared Data. The outputs are physical discrete outputs or global discrete data in Shared Data. This rung moves the inverse of the input to the output.

Syntax
RUNGMVNOT input,output

Example
Turn on a physical discrete output when the data from Scale A is invalid.

```
RUNGMVNOT s_261,p_508
```

RUNGOR

Usage
RUNGOR adds a ladder rung to the Ladder Logic. The ladder is run every 55 milliseconds in the O/S whenever there is a change in the ladder inputs. The rung inputs are physical discrete inputs or global discrete data from Shared Data. The outputs are physical discrete outputs or global discrete data in Shared Data. This rung takes two inputs, OR's them together, and outputs the value.

Syntax
RUNGOR input1,input2,output

Example
Turn on a physical discrete output if Scale A or Scale B is in motion.

```
RUNGOR s_200,s_208,p_508
```

RUNGORNOT

Usage
RUNGORNOT adds a ladder rung to the Ladder Logic. The ladder is run every 55 milliseconds in the O/S whenever there is a change in the ladder inputs. The rung inputs are physical discrete inputs or global discrete data from Shared Data. The outputs are physical discrete outputs or global discrete data in Shared Data. This rung takes two inputs, OR's them together, and outputs the inverse value.

Syntax
RUNGORNOT input1,input2,output

Example
Turn on a physical discrete output when either the JagBASIC application turns off a temporary output or a physical discrete input is turned off. The JagBASIC application must DEFSHR the s_250 global discrete data and then can toggle its value on or off.

```
RUNGORNOT  s_250,p_103,p_502
```

STARTIME

Usage
Starts the internal timer. The maximum timer value is 65 seconds.

Syntax
STARTIME milliseconds

milliseconds The time in milliseconds to start the internal timer.

Example 1

```
10 EVENT TIME
20 STARTIME 2000
30 WAITEVENT
40 IF EVENT("time") THEN PRINT "TIMER EXPIRED"
50 CLREVENT
60 GOTO 30
```

Example 2

```
10 EVENT TIME
20 ON EVENT TIME GOSUB 1000
30 STARTIME 3000
.
. MAIN PROGRAM
.
1000 PRINT "TIMER EXPIRED"
1010 CLREVENT TIME
1020 RETURN
```

STOPTIME

Usage
Stops a running timer.

Syntax
STOPTIME

Example

```
10 EVENT TIME
20 STARTIME 2000
200 STOPTIME
```

WAITEVENT

Usage
Suspends program execution until an event trigger causes program execution to resume.

Syntax
WAITEVENT

Example 1

```
10 DEFSHR EVENT SP%,s_210
20 CLREVENT
30 WAITEVENT
40 IF EVENTON("SP%")=0 THEN GOTO 20
50 IF SP%=0 THEN PRINT "ABOVE SETPOINT" ELSE PRINT "BELOW SETPOINT"
60 GOTO 20
```

Example 2

```
10 DIM SPFEED%(4)
20 DEFSHR EVENT SPFEED%(1),s_210
30 DEFSHR EVENT SPFEED%(2),s_214
40 DEFSHR EVENT SPFEED%(3),s_218
50 DEFSHR EVENT SPFEED%(4),s_21c
60 EVENT key
100 DIM DOUT(4)
110 DEFSHR DOUT(1),p_500
120 DEFSHR DOUT(2),p_501
130 DEFSHR DOUT(3),p_502
140 DEFSHR DOUT(4),p_503
200 CLREVENT
210 WAITEVENT
220 FOR i%=1 to 4
230 IF EVENTON("SPFEED%(i%)")=0 THEN GOTO 250
240 IF SPFEED%(i%)=0 THEN DOUT(i%)= 0 ELSE DOUT(i%)= 1
250 NEXT I%
260 c$=INKEY$
270 IF c$<>"" THEN GOSUB 500
280 GOTO 200
500 REM process keystroke
510 PRINT c$
520 RETURN
```

Timing Commands

JagBASIC offers several commands that work with date and time. The most fundamental timing commands, DATE$ and TIME$, simply display the current system date and time. You can also change the terminal system date and time with these commands.

Timing commands also enable your program to provide information about when or how long a certain event took place. These commands can be used to tell when a file was opened or how long it took to execute a section of code.

The SLEEP command lets you pause the program for a specified number of milliseconds. This command can be used to provide time for the user to read the screen. The program will resume execution after the time has elapsed or whenever the user presses a key.

This section discusses the following JagBASIC timing commands:

Command	Usage
CLKTICK	Allows more precise tming loops and timing of events
DATE$	Sets or returns the terminal system date.
JULDAT	Converts a date-time string: "mm-dd-yyyyHH:MM:SS" to a double precision Julian Date number.
SLEEP	Suspends program execution for the of specified number of milliseconds.
TIMDAT$	TIMDAT$ converts a double precision floating point Julian Date number to a string: "mm-dd-yyyyHH:MM:SS".
TIMER	Returns a double precision floating point number that contains the elapsed time in seconds since 00:00:00 GMT, January 1, 1970.
TIME$	Sets or returns the terminal system date and time.

TIPS

Time and Date

The Shared data variables Jag19 and Jag20 have the date and time formatted as specified in the terminal setup. These shared data variables are NOT updated automatically. However, executing either a date$ or time$ command will cause both to be updated. Here's a sample clock program:

```
5 DEFSHR CurTime, Jag20

10 DEFSHR CurDate, Jag19

15 a$=TIME$

20 PRINT left$(CurDate,6)+" "+CurTime

25 SLEEP 100

30 GOTO 15
```

CLKTICK

CLKTCK allows more precise timing loops and timing of events. CLKTICK returns a double float number that is the number of clock ticks that have occurred since the last power up of the terminal. The JAGXTREME terminal's clock ticks 36 times per second or approximately once every 27.5 seconds. When CLKTICK reaches the number 4,294,967,295, it wraps to zero. This occurs in about 3.7 years.

DATE$

Usage
Sets or returns the terminal system date.

Syntax
DATE$
DATE$="mm-dd-yyyy"

mm-dd-yyyy Month, day, and year. You do not need to enter a leading zero in front of single-digit month or day values.

Example
10 a$="10-16-1997"

20 DATE$=a$

30 PRINT DATE$

50 TIME$="10:05:00"

60 PRINT TIME$

JULDAT ()

Usage

JULDATE converts a date-time string: "mm-dd-yyyyHH:MM:SS" to a double precision Julian Date number.

The Julian Date format is a compact, numerical representation of the date and time. It is the number of seconds since 00:00:00 GMT, January 1, 1970. Since the Julian Date is numerical, it is convenient for doing mathematical computations on the date and time.

Syntax

JULDATE ("mm-dd-yyyyHH:MM:SS")

Example

10 a$ = "11-21-200010:37:00"

20 b# = JULDATE(a$)

30 PRINT "Julian Date = ",b#

Output: Julian Date = 974820420

TIMDAT$ ()

Usage

TIMDAT$ converts a double precision floating point Julian Date number to a string:

"mm-dd-yyyyHH:MM:SS".

Syntax

TIMEDAT$(Julian date)

Example

```
10 b# = 974820420
20 a$ = TIMDAT$(b#)
30 PRINT "Date and Time:":a$
```

Output: Date and Time: 11-21-200010:27:00

SLEEP

Usage
Suspends program execution for the specified number of milliseconds. The terminal timer interrupts every 27.5 milliseconds, so SLEEP can be set up to this accuracy. This command is frequently used to pause a program so the user has time to read the output screen.

Syntax
SLEEP [milliseconds]

milliseconds The number of milliseconds that you want to suspend program execution.

Example

```
10 PRINT "Taking a 10 second nap..."
20 SLEEP 10000
30 PRINT "Wake up!"
```

TIMER

Usage
Returns a double precision floating point number that contains the elapsed time in seconds since 00:00:00 GMT, January 1, 1970. Used to time the length of specific operations.

Syntax
TIMER()

Example

```
10 time#=TIMER();
20 SLEEP 1000
30 LPRINT TIMER()-time#
```

TIME$

Usage
Sets or returns the terminal system time.

Syntax
TIME$
TIME$="hh:mm:ss"

hh:mm:ss Hours, minutes and seconds.

Example
10 a$="10-16-1997"

20 DATE$=a$

30 PRINT DATE$

50 TIME$="10:05:00"

60 PRINT TIME$

Error Trapping Commands

Despite all of your efforts, errors can occur in your program. JagBASIC offers both error trapping and error handling commands for runtime errors. Runtime errors can be difficult to locate because they may occur only when a certain combination of circumstances occur. Runtime errors can also be caused by circumstances outside of your programming control, such as looking up nonexistent records in a file or accessing a remote shared data item when the Ethernet connection is down.

JagBASIC's debug commands assist you in finding runtime errors.

JagBASIC's error handling commands tell the program what to do if an error occurs. Only certain errors can be handled at run time.

JagBASIC's error commands can return an error code for the error, return the line number where the error occurred, or provide error handling instructions. Chapter 9 contains a list of JagBASIC error codes.

This section discusses the following JagBASIC error trapping commands:

Command	Usage
ERL()	Returns the line number where the error occurred, or the closest line number before the line where the error occurred
ERR()	Returns the runtime error code for the most recent error
ERROR	Simulates an occurrence of an error.
ON ERROR GOSUB	Enables error handling and, when a run time error occurs, directs your program to an error handling routine.
ON ERROR GOTO	Enables error handling and, when an error occurs, directs your program to an error handling routine.

ERL(), ERR(), ERROR

Usage
ERL returns the line number where the error occurred, or the closest line number before the line where the error occurred. Used as a debugging aid to fix runtime errors in your program.

ERR returns the runtime error code for the most recent error. Used in error handling routines to help identify the program and determine whether the program can recover from the error.

ERROR simulates an occurrence of an error. Used to debug error handling routines.

Syntax
ERL( )
ERR( )
ERROR number%

number% Error code.

Example
```
10  ON ERROR GOSUB 1000

20  ERROR 22

30  END

40  IF ERR( )=error_code THEN GOSUB 4000

.

1000 LPRINT  ERR( )

1010 LPRINT  ERL( )
```

ON ERROR GOSUB

The following errors can be trapped with the "on error" command.

File open failed	0
Resource In Use	3
Record not found	6
Device Error	13
Command Error	14
Invalid Shared Data Name	28
Shared Data String Too Long	31
No Remote Access	32

Usage
Enables error handling and when a run time error occurs the command directs the program to an error handling routine. If ON ERROR GOSUB is not used, any run time error ends the program.

Syntax
ON ERROR GOSUB line

line The first line of the error handling routine.

Example
```
10  ON ERROR GOSUB 1000

20  OPEN "X.DAT" FOR INPUT AS #1

.

.

.

1000 IF ERR()=0 THEN PRINT "FILE ERROR"

1010 PRINT "ERROR ON LINE ";  ERL( )

1020 RETURN : REM returns to the next line after error
```

ON ERROR GOTO

The ON ERROR GOTO error handling routine differs from the ON ERROR GOSUB routine in that control does not return to the next line of the program. The ON ERROR GOTO error routine must explicitly jump to the next line of the execution.

You must be particularly careful of processing errors that occur in the middle of WHILE-WEND loops, FOR-NEXT loops, and GOSUB routines. These structures create processing stacks and, if you do not clear these stacks by properly exiting these processing structures, you will eventually get an OVERFLOW error.

Usage
Enables error handling and when an error occurs directs the program to an error handling routine. If ON ERROR GOTO is not used, any run time error ends the program.

Syntax
ON ERROR GOTO line

line The first line of the error handling routine.

Example
```
10 ON ERROR GOTO 100
20 DEFSHR w#, j1/wt110
30 DEFSHR x#, j2/wt110
40 DEFSHR y#, j3/wt110
50 sum# = w# + x# + y#
60 PRINT sum#
70 GOTO 50
100 IF err() <> 32 then end
110 PRINT "JAGXTREME offline"
120 GOTO 50
```

TCP/IP Commands

JagBASIC TCP/IP "sockets" commands allow JagBASIC application programs to utilize the TCP/IP communications. A JagBASIC application can communicate to a TCP/IP application running on a PC host or to a JagBASIC application in another JAGXTREME terminal. The JagBASIC socket commands have functionality similar to the BSD socket commands. A JagBASIC application may have up to eight open sockets at once. JagBASIC closes all open sockets when the JagBASIC application terminates.

Commands	Usage
ACCEPT$	Allows new connection request to be accepted
CONNECT	Initiates TCP/IP connection to a remote host
IPD	Converts a double float representation of an IP address to a dotted string representation of an IP address
IPS	Converts the dotted string representation of an IP address to a double for storage in Shared Data.
LISTEN	Initiates TCP/IP to begin queuing connection requests
RECV$	Allows data to be received over an established connection
SEND	Allows data to be sent over an established connection
SOCKET	Creates socket for CONNECT command
SOCKCLS	Closes an established connection
SOCKOPT	Sets socket to blocking or non-blocking

TIPS

JagBASIC prints the specific socket error codes to the BAS_TERMINAL port, when you have configured this port.

TCP/IP Socket Errors

ENOBUFS	No buffers
ETIMEDOUT	Timed Out
EISCONN	Is connected
EOPNOTSUPP	Operation not supported
ECONNABORTED	Connection aborted
EWOULDBLOCK	Operation would block
ECONNREFUSED	Connection refused
ECONNRESET	Connection reset
ENOTCONN	Not connected
EALREADY	Already in error state
EINVAL	Invalid
EMSGSIZE	Bad message size
EPIPE	Socket not connected
EDESTADDRREQ	Destination address request
ESHUTDOWN	Shutdown
ENOPROTOOPT	No protocol option
EHAVEOOB	Has out of band data
ENOMEM	No memory
EADDRNOTAVAIL	Address not available
EADDRINUSE	Address in use
EAFNOSUPPORT	No support
EINPROGRESS	Operation in progress
ELOWER	Lower (IP) layer error
27. EIEIO	Bad input/output on Old McDonald's farm

TCP/IP Examples

These two sample programs illustrate how two JAGXTREME terminals to talk back-to-back using JagBASIC TCP/IP communications

```
5  REM Client Example
10 SOCK%=SOCKET()
20 stat%=SOCKOPT(SOCK%,-1)
30 stat%=CONNECT(SOCK %,"146.207.105.244",1920)
40 IF stat%=-1 THEN PRINT "connecting":SLEEP 200:GOTO 20
50 IF stat%=0 THEN PRINT "failed":stat%=SOCKCLS(sock%):SLEEP 2000:end
```

```
60 PRINT "connect success":SLEEP 2000
70 a$="Hello Dolly":len%=len(a$)
80 FOR i% = 1 TO 20
90 stat%=SEND(sock%,"Hello Dolly")
100 IF stat%<>len% THEN PRINT "send failed":SLEEP 2000:end
110 PRINT "sending ";i%:LPRINT "sending ";i%:sleep 100
120 NEXT i%
130 PRINT "closing SOCKET":sleep 2000
140 stat%=SOCKCLS (SOCK%)
150 END
```

```
10 REM Server Example
20 lsocket%=listen(1920)
30 asocket%=accept$(lsocket%)
40 IF INKEY$=chr$(2) THEN GOTO 200
50 IF asocket%=0 THEN GOTO 200
60 IF asocket%=-1 THEN PRINT "awaiting connect":SLEEP 100:GOTO 30
70 PRINT "ip";IP$:LPRINT "ip=";IP$:SLEEP 2000
80 i%=0
90 a$=RECV$(asocket%,11)
100 IF INKEY$=chr$(2) then GOTO 200
110 IF a$="" then sleep 100:GOTO 90
120 i%=i%+1:PRINT "receiving ";i%:LPRINT i%;" ";a$
130 IF i% < 20 THEN GOTO 90
200 PRINT "closing socket":SLEEP 2000
210 stat%=SOCKCLS(lsocket%):stat%=SOCKCLS(asocket%)
220 END
```

ACCEPT$

Usage

ACCEPT$ allows the JagBASIC application to accept new connection requests that remote clients are initiating. The JagBASIC application must supply an integer number that is the socket number of the LISTEN connection. If ACCEPT$ finds a new connection, it creates a new socket for the new connection.

The ACCEPT$ command may be either blocking or non-blocking. The default is non-blocking. When in non-blocking mode, the ACCEPT$ command returns a status indicating whether it has accepted a new connection request. If there is no new connection, the JagBASIC application must periodically issue the ACCEPT$ command to know when a new connection occurs.

If the socket is in blocking mode, ACCEPT$ suspends execution until it has accepted a connection from a remote host. The JagBASIC application uses the SOCKOPT command to put the socket in blocking or non-blocking mode.

The return value is an integer variable. If it is successful, the ACCEPT$ returns the socket number of the new connection. If there is no new connection, ACCEPT$ returns a (–1). If there is a fatal error, ACCEPT$ returns a (0).

When there is a successful new connection, ACCEPT$ also sets the IP address of the remote node that initiated the connection in the variable IP$.

Syntax
newSocket% = ACCEPT$(socket%)

Example
10 REM Server Example

20 lsocket%=listen(1920)

30 asocket%=accept$(lsocket%)

CONNECT

Usage
The CONNECT function initiates a TCP/IP connection to a remote host. The JagBASIC application must supply the socket number, the host IP address string, and the host port integer.

The connection attempt may be a blocking or a non-blocking attempt. The default is non-blocking. When it is a non-blocking, CONNECT returns an "in progress" status when the connection is in progress but has not yet completed. Then, the JagBASIC application must periodically re-issue the CONNECT to know when the connection is complete. If it is a blocking attempt, JagBASIC suspends execution until a successful connection is made. The JagBASIC application uses the SOCKOPT command to put the socket in blocking or non-blocking mode.

The return value is an integer variable. If the connection attempt is successful, CONNECT returns a (1). If the connection is still "in progress" but not completed, it returns a (-1). If the connection attempt fails, CONNECT returns a (0).

Syntax
status% = CONNECT(socket%, IPAddress$, hostport%)

Example
5 REM Client Example

10 sock%=SOCKET()

20 stat%=SOCKOPT(sock%,-1)

30 stat%=CONNECT(sock%,"146.207.105.244",1920)

IPD

Usage

The IPD function converts the dotted string representation of an IP address to a double for storage in Shared Data. If successful, "ipd" returns the IP address. If not successful, ipd returns a 0.

Syntax

Ipadr# = ipd(ipstring$)

Example

```
5  DEFSHR subnetmask#,net03
10 subnetmask# = IPD ("255.255.255.0")
```

IPS

Usage

The IPS$ function converts a double float representation of an IP address to a dotted string representation of an IP address, e.g., "111.111.111.123". Use it for converting an IP address that is retrieved from Shared Data to its string representation.

Syntax

A$ = IPS$(ipdouble#)

Example

```
5  defshr ipaddress#,net02
10 ipaddress$ = ips$(ipaddress#)
```

LISTEN

Usage
LISTEN function initializes TCP/IP to begin queuing the connection requests for the host port. The JagBASIC application must supply an integer host port number. Subsequently, the ACCEPT$ command allows the JagBASIC application to begin accepting the connection requests from a remote node. Remote clients initiate the connection requests with the CONNECT command.

The return value is an integer variable. If it is successful, LISTEN returns the socket number. If LISTEN fails, it returns a (0).

Syntax
socket% = LISTEN(hostport%)

Example

```
10 REM Server Example
20 lsocket%=LISTEN(1920)
30 asocket%=ACCEPT$(lsocket%)
```

RECV$

Usage
RECV$ command allows the JagBASIC to receive data over an established connection. The JagBASIC application must supply an integer socket number and length for the received

string. The maximum received data length on each call is the JagBASIC maximum string size (160 bytes).

The RECV$ command may be blocking or non-blocking. The default is non-blocking. When it is in non-blocking mode, RECV$ returns data immediately or returns a status indicating there is no data available. In this mode, the JagBASIC application must periodically re-issue the RECV$ command to see if there is more data. If the socket is in blocking mode, RECV$ suspends execution until the socket receives data. The JagBASIC application uses the SOCKOPT command to put the socket in blocking or non-blocking mode.

The return value is a string variable. If it is successful, RECV$ returns the data string. If there is no data available on the connection, RECV$ returns the null string. If there is a fatal error on receiving, RECV$ sets the JagBASIC "device error". The JagBASIC application must use the ON ERROR GOTO or ON ERROR GOSUB statements to trap this error.

Syntax
inputString$ = RECV$(socket%, length%)

Example
```
10 REM Server Example
20 lsocket%=LISTEN(1920)
30 asocket%=ACCEPT$(lsocket%)
40 IF INKEY$=chr$(2) THEN GOTO 200
50 IF asocket%=0 then GOTO 200
60 IF asocket%=-1 then print "awaiting connect":sleep 100:GOTO 30
70 PRINT "ip";IP$:LPRINT "ip=";IP$:sleep 2000
80 i%=0
90 a$=RECV$(asocket%,11)
```

SEND

Usage
The "send" command allows the JagBASIC to send data over an established connection. The JagBASIC application must supply an integer socket number and the string to be sent.

The return value is an integer variable. If SEND is successful, it returns a positive number that is the number of characters sent. If it fails, SEND returns a (0).

If there is a fatal error on sending, SEND sets the JagBASIC "device error". The JagBASIC application must use the ON ERROR GOTO or ON ERROR GOSUB statements to trap this error.

Syntax
numChars% = SEND(socket%, stringToSend$)

Example
```
5   REM Client Example
10 sock%=SOCKET()
20 stat%=SOCKOPT(sock%,-1)
30 stat%=CONNECT(sock%,"146.207.105.244",1920)
```

```
40 IF stat%=-1 THEN PRINT "connecting":SLEEP 200:GOTO 20
50 IF stat%=0 THEN PRINT "failed":stat%=sockcls(sock%):SLEEP 2000:END
60 PRINT "connect success":SLEEP 2000
70 a$="Hello Dolly":len%=LEN(a$)
80 FOR i% = 1 to 20
90 stat%=SEND(sock%,"Hello Dolly")
```

SOCKET

Usage
The SOCKET function creates a socket for a subsequent CONNECT command, which initiates a connection to a remote host using this socket.

The return value is an integer variable. If it is successful, SOCKET returns the socket number. If it fails, SOCKET returns a (0).

Syntax
socket% = SOCKET( )

Example

```
5  REM Client Example
10 sock%=SOCKET()
20 stat%=SOCKOPT(sock%,-1)
```

SOCKCLS

Usage
SOCKCLS command allows the JagBASIC application to close an established TCP/IP connection. The JagBASIC application must supply an integer number. SOCKCLS returns an integer 1.

Syntax
stat% = SOCKCLS(socket#)

Example

```
130 PRINT "closing socket":SLEEP 2000
140 stat%=SOCKCLS(sock%)
150 END
```

SOCKOPT

Usage

The SOCKOPT function makes a TCP/IP socket blocking or non-blocking. The default is non-blocking. The JagBASIC application must supply an integer socket number and an integer option number. If the option is 1, SOCKOPT makes the socket a blocking socket. If the option is –1, it makes the socket a non-blocking socket. The blocking/non-blocking functionality applies only to the ACCEPT$, LISTEN, and RECV$ commands.

The return value is an integer variable. If the command is successful, it returns a (1). If the command fails, it returns a (0).

Syntax

status% = SOCKOPT(socket%, option%)

Example

```
5  REM Client Example

10 sock%=SOCKET()

20 stat%=SOCKOPT(sock%,-1)
```

6

Shared Data Variables

The shared data database is the main data storage area for JAGXTREME information. This central variable table keeps track of virtually every data value used by the JAGXTREME terminal. All operating system tasks can directly use these shared values.

The Scale threads or Setup is the main shared data source. Other "external" agencies such as JagBASIC, the Windows API, Save/Restore/Setup Utility, Allen-Bradley interface, MODBUS Plus interface, or PROFIBUS interface can also read or write to shared data.

External write access to shared data variables is sometimes restricted. The Scale threads and Setup maintain write access to any variable, but access to Setup itself may be restricted by the Legal-for-Trade jumper. If the Legal-for-Trade jumper is installed, any shared data variables listed, as "External Read Only" cannot be written to by external agencies. If the Legal For Trade jumper is removed, there are no external write access restrictions. Access restrictions are enforced on a whole block basis only. Note: Dipswitch 1 must be on for setup to be entered.

See the JAGXTREME Operating Environment and Shared Data and the Shared Data Types sections in Chapter 1 for more information.

This chapter lists the various shared data variables. The following abbreviations are used throughout the chapter.

- **UC**—Unsigned Character
- **C**—String Character variables are any ASCII characters with values in the range 1 to 127 or extended characters in the range 128 through 255, terminated by a 0.
- **D**—Double Float variables are numeric variables in 64-bit double-precision format.
- **L**—Long variables are numeric integers representing a number of eight or more digits.
- **US bit**—Unsigned bit variables have a value of 0 or 1.

Shared Data Heap Elements

This section lists the shared data heap elements. These variables hold the values associated with different scale weights and with board configurations.

Scale Weight Shared Data

These variables hold the shared data values associated with scale weight. The fields are **external read only**. The '**n**' listed below in Local Field ID will be replaced with the internal scale number. The scale number can be from 1 to 5.

Local Method	Local File Id	Internal Format	External Format
DisplayedGrossWeight	/wt**n**01	12C	12 alphanumeric, right-justified
DisplayedNetWeight	/wt**n**02	2C	12 alphanumeric, right-justified
DisplayedWeightUnits	/wt**n**03	2C	2 alphanumeric (**lb**pounds, **kg**kilograms, **g**rams, **t**metrictons)
DisplayedAuxGrossWeight	/wt**n**04	12C	12 alphanumeric, right-justified
DisplayedAuxNetWeight	/wt**n**05	12C	12 alphanumeric, right-justified
DisplayedAuxWeightUnits	/wt**n**06	6C	6 alphanumeric (**lb**pounds, **kg**kilograms, **oz**ounces, **lb-oz**pounds & ounces, **ozt**roy ounces, **dwt**penny weights, metric **t**ons, **ton**, or custom units name)
DisplayedAuxRatePeriod	/wt**n**07	C	1 alphanumeric (**N**o, **S**ec, **M**in, **H**our)
DisplayedRate	/wt**n**08	12C	12 alphanumeric, right-justified
DisplayedDiagnosticWeight	/wt**n**09	12C	12 alphanumeric, right-justified
LegalGrossWeight	/wt**n**10	D	double float weight
LegalNetWeight	/wt**n**11	D	double float weight
AuxiliaryGrossWeight	/wt**n**12	D	double float weight
AuxiliaryNetWeight	/wt**n**13	D	double float weight
AuxiliaryRate	/wt**n**14	D	double float weight
ScaleState	/wt**n**15	UC	0=disabled, 1=normal weight processing, 2=diagnostic, 3=calibration, 4=shift adjust.
ContinuousOutputStatusA	/wt**n**16	UC	1 byte, any value.
FineGrossWeight	/wt**n**17	D	double float weight
FineNetWeight	/wt**n**18	D	double float weight
Weighing Range	/wt**n**19	UC	0= single weighing range, 1=multi-range 1, 2=multi-range 2, 3=multi-range 3
WIM Time Counts	/wtn20	D	Double float WIM time counts
WIM Weight Units	/wtn21	3C	3 alphanumeric (**lb**pounds, **kg**kilograms, **g**rams, **t**metric tons)

Board Configuration Shared Data

These variables hold the shared data values associated with board configuration. Board configuration shared data variables are initialized at power up. The fields are **external read only**.

Local Method	Local File Id	Internal Format	External format
Read the JAGXTREME display	/bd001	50C	Read contents of JAGXTREME display. The first 8 bytes have the contents of the numeric display. The next 16 bytes have the contents of the alphanumeric dislay. The next 8 bytes have the contents of thenumeric cursors. The next 16 bytes have the contents of the alphanumeric cursors. For each cursor, its corresonding byte has an ASCII "O" if the cursor is off and an ASCII '1' if the cursor is on.
Board configuration string	/bd002	60C	Contains a 15-byte entry for each of four board slots. Each entry contains a two-character board identifier and a 13-character board software serial number where applicable.
Latest keystroke/key source	/bd003	2C	2 alphanumeric; read/write
EEPROMAutorizationByte	/bd004	10C	10 alphanumeric; read only
ConsoleSoftwarePartNo	/bd005	13C	12 alphanumeric; read only
Scale1SoftwarePartNo	/bd006	13C	12 alphanumeric; read only
Scale2SoftwarePartNo	/bd007	13C	12 alphanumeric; read only
Scale3SoftwarePartNo	/bd008	13C	12 alphanumeric; read only
Scale4SoftwarePartNo	/bd009	13C	12 alphanumeric; read only
Scale5SoftwarePartNo	/bd0010	13C	12 alphanumeric; read only
MultiFunctionIOSoftwarePartNo	/bd011	13C	12 alphanumeric; read only
POWERCELL SoftwarePartNo	/bd012	13C	12 alphanumeric; read only
DisplayContents	/bd013	17C	Reserved for JAGXTREME O/S Use Only

HMI Weight Stream	/bd014	105C	Contains HMI weight streams for up to five scales. The HMI subscribes to which fields the JAGXTREME will send by sending the subscribe message. Its format is: <STX>S<ABCDESJL<ETX><chk> where ABCDE represents the scales, S represents the selected scale, L reprsents the lower JAGXTREME display, and J is the JagBASIC message filed. "S" is mutually exclusive from ABCDE. Jag60 stores subscription. The HMI weight stream is formatted as follows: Stream1><US><stream2><US><stream n> Each weight stream has the following contents: <JagX ID> 1N Range: 1 to 6 <Scale ID> 1A Range: A to E If selected scale, range is in lowercase <a to e>. <Status> 1C Bit 7 Always 0 Bit 6 Always 1 Bit 5 1=Scale in Motion Bit 4 1 = Center of Zero Bit 3-2 00=single range 01=weight range 1 02 = weight range 2 03 = weight range 3 Bit 1 1 = Net Mode Bit 0 1 = Present Tare <Units> 1 N 0=None, 1=lb, 2=kg, 3=g, 4=oz. 5=ozt, 6=dwt. 7=t. 8=ton, 9=custom <Net Wt> 8N 6 digits plus ossible "_" and "." <Tare Wt> 8N 6 digits plus possible "_" and "."
PowerCellScale-CellErrors	/bd016	25C	25 Bytes. There is an error number for up to 24 POWERCELLS. This field has cell errors for both Scale A, Scale B, Scale C, and Scale D.

PowerCellScale_CellCounts	/bd017	24D	192 Bytes. Each double float contains the current shift-adjusted counts for consecutive power cells in a scale. An external agency can request the current count for a scale by setting trigger /t_69d for Scale A, /t_6ad for Scale B to 1, /t_62d for Scale C, and /t_63d for Scale D.
ScanTable	/bd018	25C	Scan Table contains ordered list of current power cell addresses.
PowerCellScale-CellCounts	/bd019	24L	96 bytes. Each long contains the current shift-adjusted coutns for consecutive POWERCELLs in a scale. An appliation can request the current counts for a scale by setting trigger t_69d for Scale A, t_6ad for Scale B to1, t_62d for Scale C, and t_63d for Scale D. Read only.
PowerCellOverloadState	/bd020	25C	25 Bytes. There is one entry each for up to 24 power cells. 0 = Cell not assigned, 1 = Cell OK, 2 = Cell in Overload condition.
PowerCellZeroDriftState	/bd021	25C	25 bytes. There is one entry each for up to 24 POWERCELLs. 0=Cell not assigned, 1=Cell OK, 2= Cell in Overload condition.
Read Discrete Inputs	/bd030	US	Retrieves the status of all discrete inputs p_100 through p_10f
Read Discrete Outputs	/bd031	US	Retrieves the status of all discrete outputs p_500 through p_50f
Read Status Flags for Scale A	/bd032	US	Retrieves scale A status bits s_200 through s_207 and s_260 through s_264
Read Status Flags for Scale B	/bd033	US	Retrieves scale B status bits s_208 through s_20f and s_268 through s_26c
Read Status Flags for Scale C	/bd034	US	Retrieves scale C status bits s_270 through s_27c
Read Status Flags for Scale D	/bd035	US	Retrieves scale D status bits s_280 through s_28c
Read Status Flags for Scale E	/bd036	US	Retrieves scale E status bits s_2f0 through s_2fc
Read JagBASIC Custom Flags	/bd037	US	Retrieves custom status bits s_250 through s_25f
Display Board	/bd085	US bit	1=Yes, 0=No; Read only.
AnalogBoard1	/bd086	US bit	1=Yes, 0=No; Read only.
AnalogBoard2	/bd087	US bit	1=Yes, 0=No; Read only.
AllenBradleyPLC	/bd088	US bit	1=Yes, 0=No; Read only.
PROFIBUS	/bd089	US bit	1=Yes, 0=No; Read only.
Ethernet	/bd090	US bit	1=Yes, 0=No; Read only.
MultiFunctionIO1	/bd091	US bit	1=Yes, 0=No; Read only.
PowerCell	/bd092	US bit	1=Yes, 0=No; Read only.
ModBus Plus	/bd093	US bit	1=Yes, 0=No; Read only.
AnalogOut	/bd094	US bit	1=Yes, 0=No; Read only.
HighPrec1	/bd095	US bit	1=Yes, 0=No; Read only.
HighPrec2	/bd096	US bit	1=Yes, 0=No; Read only.
Multi-FunctionIO2	/bd097	US bit	1=Yes, 0=No; Read only.

Shared Data Static RAM Elements

This section lists the shared data static random access memory elements. These elements include variables for scale weight, scale calibration parameters, scale tare weight, setpoints, system values, user literals, user prompts, user variables, cluster variables, PLC configuration, templates, security, serial port setup, network interface, network remote nodes, network host workstation nodes, analog output, connections, ladder logic, and BASIC applications. These fields are preserved when the terminal is powered down.

Scale Weight Stored in Static RAM Shared Data

These shared data variables hold the values associated with scale weight stored in static RAM. The fields are external read only. The 'n' will be replaced with the Internal Scale number. The scale number can be from 1 to 5.

Local Method	Local File Id	Internal Format	External Format
ScaleModeOut	/ws**n**01	C	1 alphanumeric (**G**ROSS or **N**ET)
DisplayedTareWeight	/ws**n**02	12C	12 alphanumeric, right-justified
DisplayedAuxTareWeight	/ws**n**03	12C	12 alphanumeric, right-justified
FineTareWeight	/ws**n**04	D	double float weight
AuxiliaryTareWeight	/ws**n**05	D	double float weight
CurrentUnits	/ws**n**06	UC	1=Primary, 2=Secondary
TareSource	/ws**n**07	UC	1=Pushbutton, 2=Keyboard, 3=Autotare
CurrentZeroCounts	/ws**n**08	D	Double PB&AZM current zero counts
TareSourceString	/ws**n**09	2C	"PT"=keyboard tare, else "T"
DisplayedStoredWeight	/ws**n**10	12C	12 A/N, right justified
Stored Weight	/ws**n**11	D	double float weight
LegalTareWeight	/ws**n**12	D	double float weight
LastScaleError	/ws**n**13	41C	Date – time - error message
NumberScaleErrors	/ws**n**14	F	Errors since calibration or reset

Scale Calibration Parameters Stored in Static RAM

These shared data variables hold the values associated with scale calibration parameters stored in static RAM. The fields are **external read only**. The '**n**' will be replaced with the Internal Scale number.

Local Method	Local File Id	Internal Format	External Format
AuxiliaryDisplayUnits	/cs**n**01	UC	**1=**pounds, **2=**kilograms, **3=**grams, **4=**ounces, **5=**pounds & ounces, **6=**troy ounces, **7=**penny weights, **8=**metric tons, **9=**tons, **10=**custom units
CustomUnitsName	/cs**n**02	6C	6 alphanumeric
CustomUnitsConversionFactor	/cs**n**03	D	double float
RateIntegrationPeriod	/cs**n**04	C	1 alphanumeric (**N**o, **S**ec, **M**in, **H**our)
RateSampleTime	/cs**n**05	UC	seconds
RateDisplayFrequency	/cs**n**06	UC	0=every second,1=every five seconds,2=every half second
IDNET Higher Precision	/cs**n**08	UC	0=Normal 1=Higher
PowerUpTimer	/cs**n**09	UC	2 alphanumeric, right-justified (in minutes)
LowPassFilterCornerFrequency	/cs**n**10	D	double float (0.1 Hz to 9.9 Hz in steps of 0.1 Hz)
NotchFilterFrequency	/cs**n**11	D	double float (0.1 Hz to 9.9 Hz in steps of 0.1 Hz)
CombFilterFrequency	/cs**n**12	D	double float (0.1 Hz to 9.9 Hz in steps of 0.1 Hz)
PrintThreshold	/cs**n**13	D	double float weight
PrintResetThreshold	/cs**n**14	D	double float weight
DisplayUpdateFrequency	/cs**n**15	D	double float hertz
CustomContinuousOutUpdateFreq	/cs**n**16	D	double float hertz
LowPassFilterPoles	/cs**n**17	US bit	unsigned integer.
ScaleID	/cs**n**18	8C	8 bytes text string.
AveragingFilterOrder	/cs**n**19	US bit	unsigned integer.
CombFilterOrder	/cs**n**20	US bit	unsigned integer.
ScaleType	/cs**n**21	C	1 alphanumeric (**A**nalog Load Cells, **P**ower Digital Load Cells, **I**High Precision, **S**ingle cell DigiTOL, Power **M**odule DigiTOL, UltraRes**H**igh, or UltraRes**L**ow)
ScaleLocation	/cs**n**22	UC	0=first unit, 1=second unit (board or COM: port)
IDNetVibrationAdaptor	/cs**n**23	C	'0' - '9' (specific to Precision Base)
IDNetWeighingProcessAdaptor	/cs**n**24	C	'0' - '9' (specific to Precision Base)
IDNetAutomaticStabilityDetection	/cs**n**25	C	'0' - '9' (specific to Precision Base)
IDNetAutoZeroSetting	/cs**n**26	C	'0'="Off", '1'="On"
IDNetSoftwarePartNum	/cs**n**27	11C	xxxx-x-xxxx string from Precision Base
IDNetIdentcode	/cs**n**28	2C	'' to '99' calibration count from Precision Base
ScalesInSummingScale	/cs**n**29	UC	Add Scale to Summing Scale, 0=No, 1=Yes
CalibrationDate	/cs**n**30	11C	11 alphanumeric
FilInoiseFilterEnable	/cs**n**85	US bit	1=True, 0=False
AutoPrint	/cs**n**86	US bit	1=True, 0=False
NoMotionBeforePrint	/cs**n**87	US bit	1=True, 0=False
DisplayRate	/cs**n**88	US bit	1=True, 0=False
DisplayAuxiliaryUnits	/cs**n**89	US bit	1=True, 0=False
UnitsSwitchEnable	/cs**n**90	US bit	1=True, 0=False
PrintInterlockEnable	/cs**n**91	US bit	1=True, 0=False
Do_IDNET_TareInJag	/cs**n**92	US bit	1=True, 0=False
ProcessApplication	/cs**n**93	US bit	1=True, 0=False

Scale Tare Shared Data

These shared data variables hold the values associated with scale tare weight. The fields are **external read only**. The '**n**' will be replaced with the Internal Scale number. The scale number can be from 1 to 5.

Local Method	Local File Id	Internal Format	External Format
AutoTareThreshold	/tr**n**01	double	double float weight
AutoTareResetThreshold	/tr**n**02	double	double float weight
AutoClearTareThreshold	/tr**n**03	double	double float weight
TareEnabled	/tr**n**85	US bit	1=True, 0=False
PushbuttonTare	/tr**n**86	US bit	1=True, 0=False
KeyboardTare	/tr**n**87	US bit	1=True, 0=False
AutoTare	/tr**n**88	US bit	1=True, 0=False
AutoTareCheckMotion	/tr**n**89	US bit	1=True, 0=False
AutoClearTare	/tr**n**90	US bit	1=True, 0=False
AutoClearTareAfterPrint	/tr**n**91	US bit	1=True, 0=False
AutoClearTareMotion	/tr**n**92	US bit	1=True, 0=False
TareInterlock	/tr**n**93	US bit	1=True, 0=False
DisplayTare	/tr**n**94	US bit	1=True, 0=False
NetSignCorrection	/tr**n**96	US bit	1=True, 0=False

Setpoint Shared Data

These shared data variables hold the values associated with setpoints. Although the terminal's setpoints are numbered 1-12, they are referenced with an internal setpoint number 1-C, where A=10, B=11, and C=12. The fields are external read/write. The "n" will be replaced with the Internal Setpoint number (1-C).

The Setpoint Target Variable (spn03) can be used to select the type of setpoint operation required:

Gross = gross setpoint without auto preact adj

H = gross setpoint with auto preact adj

Jog = Jog Setpoint

Learn = Learn Jog Setpoint

Net = net setpoint without auto preact adj

M = net setpoint with auto preact adj

Displayed = displayed setpoint

Rate = rate setpoint

The operation of Setpoint Preact Value (spn06) will vary depending on the selection of the Setpoint Target Variable (spn03). If G, N, or D is selected, the Setpoint Preact Value (spn06) is a double float weight value, and there is no auto preact adjustment. If H or M is selected, the Setpoint Preact Value (spn06) is a double float seconds value, and auto preact adjust is enabled. As the scale is used for weighments, the terminal's operating system will adjust the time value stored in this field. When R is selected, there is no associated preact value.

The following four fields are secondary inputs to a single setpoint that has auto-adjusting preacts based on flow rates. When the flow rate is greater than threshold 3, preact 3 is used. When the flow rate is greater than threshold 2, preact 2 is used.

Threshold 3 is the higher threshold rate. Threshold 2 is the lower threshold rate. If the flow rate is below both thresholds, the standard preact is used.

Local Method	Local File Id	Internal Format	External Format
AutoAdjustSetpointThreshold2	/spc08	D	double float weight
AutoAdjustSetpointPreact2	/spc06	D	double float weight
AutoAdjustSetpointThreshold3	/spb08	D	double float weight
AutoAdjustSetpointPreact3	/spb06	D	double float weight

Latching of the setpoint is controlled by Setpoint Latching (spn87). When an external agency enables "Feed Latching", the terminal's O/S sets the Setpoint Latched=1 and the Setpoint Feeding=0 condition again until the external agency resets the Setpoint Latched=0. The external agency must reset Setpoint Latched=0 before starting a new setpoint. Any time you wish to change a setpoint value, setting, or latch, Restart Setpoints A (t_698) or Restart Setpoints B (t_6a8) must be triggered by setting its value equal to 1 in order to instruct the terminal's O/S to use the new setpoint settings.

Jog Tables for the Jog setpoints are contained in the Cluster Variable fields. The fields contain numbers in string format. Cluster variables 1-10 are the weight values. Cluster Variables 11-20 are the timer values associated with each of the weight values. The weight and timer values must be ordered in ascending order. A weight value of 0 indicates the termination of the table values.

Local Method	Local File Id	Internal Format	External Format
SetpointName	/spn01	8C	8 alphanumeric
SetpointEnbleButton	/spn02	UC	alphanumeric (0=disabled; Scale A=1, Scale B= 2, Scale C= 3, Scale D= 4, Scale E= 5)
SetpointTargetVariable	/spn03	C	1 alphanumeric (G,H,N,M,D,R,L or J)
SetpointCoincidenceValue	/spn05	D	double float weight; For learn Jog setpoints, this field contains a time value in seconds
SetpointPreactValue	/spn06	D	double float weight or double float seconds
SetpointDribbleValue	/spn08	D	double float weight
SetpointToleranceValue	/spn10	D	double float weight
SetpointFillOrDischarge	/spn86	US bit	1=Discharge, 0=Fill
SetpointLatching	/spn87	US bit	1=Feed Latching Enabled, 0=Feed Latching Disabled
SetpointLatched	/spn88	US bit	1=Latched, 0=Unlatched

System Shared Data

These shared data variables hold the values associated with system data, such as the system date and time. The fields are **external read only**.

Local Method	Local File Id	Internal Format	External Format
Current Selected Scale	/jag01	2C	First Char= L or n, 2nd=A or B
ETHERNET Node Address	/jag02	UC	8 bit address
Market	/jag04	C	1 alphanumeric (**U**SA, **E**uropean Community, **A**ustralia, **C**anada)
DateFormat	/jag05	UC	1 byte integer
TimeFormat	/jag06	UC	1 byte integer
JulianDate	/jag07	8C	8 alphanumeric
JulianTime	/jag08	8C	8 alphanumeric
Consecutive Number	/jag09	L	long integer counter
Error Message	/jag10	41C	Date – time – error message
SoftwareID	/jag11	12C	12 alphanumeric
SoftwareSerialNumber	/jag12	12C	12 alphanumeric
BRAMVersionNumber	/jag14	L	4 byte integer
NumberOfInternalScales	/jag15	UC	1 byte unsigned integer
DateSeparator	/jag16	C	1 byte character
TimeSeparator	/jag17	C	1 byte character
ConsecutiveNumberDest	/jag18	10C	Size of J_FNAME + 1
CurrentDate	/jag19	11C	11 alphanumeric
TimeOfDay	/jag20	11C	11 alphanumeric
WeekDay	/jag21	10C	10 alphanumeric
ConsecutiveNumberPreset	/jag22	L	CN Preset value
CharacterSet	/jag23	UC	0=USA, 1=France, 2=England, 3=Germany, 4=Den mark-I, 5=Sweden, 6=Italy, 7=Spain-I, 8=Japan, 9=Nor way, 10=Denbmark-II, 11=Spain-II, 12=Latin America
Language	/jag24	UC	0=English, 1=French, 2=German, 4=Spanish
Keyboard	/jag25	UC	0=English, 1=French, 2=German, 4=Spanish
Disable Memory Key	/jag91	US bit	1=True, 0=False
Error Log Reset Time	/jag26	24C	Date-Time
KeyBeeperEnable	/jag85	US bit	1=On, 0=Off
AlarmBeeperEnable	/jag86	US bit	1=On, 0=Off
LegalForTrade	/jag88	US bit	1=True, 0=False
ConsecutiveNumberEnable	/jag89	US bit	1=True, 0=False
ConsecutiveNumberPresetEnable	/jag90	US bit	1=True, 0=False
Disable Memory Key	/jag91	US bit	1=True, 0=False

User Literals Shared Data

These shared data variables hold the values associated with user literal data. The fields are **external read/write**.

Local Method	Local File Id	Internal Format	External Format
User Literal 1	/lit01	40C	40 alphanumeric
User Literal 2	/lit02	40C	40 alphanumeric
User Literal 20	/lit20	40C	40 alphanumeric

User Prompts Shared Data

These shared data variables hold the values associated with user prompts. The fields are **external read/write**.

Local Method	Local File Id	Internal Format	External Format
User Prompt 1	/pmt01	16C	16 alphanumeric
User Prompt 2	/pmt02	16C	16 alphanumeric
User Prompt 20	/pmt20	16C	16 alphanumeric

User Variables Shared Data

These shared data variables hold the values associated with user variable data. The fields are **external read/write**.

Local Method	Local File Id	Internal Format	External Format
User Variable 1	/var01	47C	USER_VARIABLE structure
User Variable 2	/var02	47C	USER_VARIABLE structure
User Variable 20	/var20	47C	USER_VARIABLE structure
VariablesInUse	/var81	UC	Number 0-20
PromptLoopingMode	/var82	UC	0=No Loop, 1=Loop

Cluster Variable Shared Data

These shared data variables hold the values associated with cluster variable data. The fields are **external read/write**. Cluster Variable fields may contain Jog Tables for the Jog Setpoints. The fields have numbers in string format. Cluster variables 1-10 are the weight values. Cluster variables 11-20 are the associated timer values.

Local Method	Local File Id	Internal Format	External Format
Cluster Variable 1	/clv01	40C	40 alphanumeric
Cluster Variable 2	/clv02	40C	40 alphanumeric
Cluster Variable 20	/clv20	40C	40 alphanumeric

Template Shared Data

These variables hold the values associated with template shared data. The fields are **external read only**.

Local Method	Local File Id	Internal Format	External Format
Printer Template 1	/ptp01	409C	400 a/n grammar + 8 a/n template name + null
Printer Template 2	/ptp02	409C	400 a/n grammar + 8 a/n template name + null
Printer Template 3	/ptp03	409C	400 a/n grammar + 8 a/n template name + null
Printer Template 4	/ptp04	409C	400 a/n grammar + 8 a/n template name + null
Printer Template 5	/ptp05	409C	400 a/n grammar + 8 a/n template name + null

Serial Port Setup Shared Data

These variables hold the values associated with serial port setup shared data, such as the transmit and receive baud rates. The fields are **external read only**. The '**n**' will be replaced with the Internal Scale number.

Local Method	Local File Id	Internal Format	External Format
InterfaceType:	/sr**n**01	UC	0=RS232, 1=RS422, 2=RS485
XmitBaudRate	/sr**n**02	UC	0=300, 1=600, 2=1200, 3=2400, 4=4800, 5=9600, 6=19200, 7=38400, 8=57600, 9=76800, 10=115200.
Parity	/sr**n**04	UC	Same as BIOS values. 0=even, 16=odd, 64=none
FlowControl	/sr**n**05	UC	Same as BIOS values. 0=none, 1=Xon/Xoff, 2=RS232.
Data Bits	/sr**n**07	UC	Same as BIOS values. 8=7 bits, 12=8 bits
Stop Bits	/sr**n**08	UC	Same as BIOS values. 1=1, 2=1.5, 3=2
Checksum	/sr**n**85	US bit	1=On, 0=Off

Network Interface Shared Data

These variables hold the values associated with network interface shared data.

Local Method	Local File Id	Internal For mat	External Format
NetworkConsole	/net91	US bit	1=True, 0=False

Network Remote Node Shared Data

These variables hold the values associated with network shared data. The fields are **external read only**. The '**n**' will be replaced with a remote node index number (1-6).

Local Method	Local Field	Internal For mat	External Format
RemoteConnectionEnabled	/rm**n**87	US bit	1=True, 0=False

Network Host Workstation Node Shared Data

These variables hold the values associated with network host workstation node shared data. The fields are **external read only**. The '**n**' will be replaced with a remote node index number (1-3).

Local Method	Local File Id	Internal Format	External Format
RemoteConnectionEnabled	/rw**n**87	US bit	1=True, 0=False

PLC Configuration on Shared Data

These variables hold the values associated with PLC configuration shared data, such as the number of scales. The fields are **external read only**.

Local Method	Local File Id	Internal Format	External Format
RackAddress	/abc01	UC	Allen-Bradley 0-59, PROFIBUS station ID 1-127, MODBUS Plus 1-63.
AllenBradleyStartingQuarter	/abc02	UC	1-4
AllenBradleyDataRate	/abc03	UC	0=57.6k, 1=115.2k, 2=230.4k
NbrOfScales	/abc05	UC	1-4
DiscreteDataFormat	/abc06	UC	0=Integer Weight, 1=Increments, 2=Extended Weight, 4=Floating Point
InputRotation	/abc07	10C	10 character string
AllenBradleyLastRack	/abc85	US bit	1=Yes, 0=No
BlockTransferEnable	/abc86	US bit	1=Yes, 0=No
ModbusPlusGlobalsEnable	/abc87	US bit	1=Yes, 0=No
PLC_ControlsScaleASetpoints	/abc88	US bit	1=Yes, 0=No
PLC_ControlsScaleBSetpoints	/abc89	US bit	1=Yes, 0=No

PLC Scale Configuration Shared Data

These variables hold the values associated with PLC configuration shared data, such as the scale location. The fields are **external read only**. The '**n**' will be replaced with a scale index number.

Local Method	Local File Id	Internal Format	External Format
TerminalNodeName	/ab**n**01	2C	2 alphanumeric (J1, J2, J3, J4, J5, J6)
ScaleSelection	/ab**n**02	UC	1 byte unsigned integer
ScaleLocation	/ab**n**85	US bit	0=Local, 1=Remote

Analog Output Shared Data

These variables hold the values associated with analog output shared data. The fields are **external read only**. The '**n**' will be replaced with a channel index number.

Local Method	Local File Id	Internal Format	External Format
AnalogOutSourceData	/ao**n**01	C	G=Gross Weight Scale 1, H=Gross Weight Scale 2, I=Gross Weight Scale 3, J=Gross Weight Scale 4, K=Gross Weight Scale 5, L=Net Weight Scale 1, M=Net Weight Scale 2, N=Net Weight Scale 3, O=Net Weight Scale 4, P=Net Weight Scale 5, Q=Rate Scale 1, R=Rate Scale 2, S=Rate Scale 3, T=Rate Scale 4, U=Rate Scale 5, B=JagBASIC Scale 1, C=JagBASIC Scale 2, D=JagBASIC Scale 3, E=JagBASIC Scale 4, F=JagBASIC Scale 5.
AnalogOutZeroTrim	/ao**n**02	D	Zero Adjustment Offset
AnalogOutSpanTrim	/ao**n**03	D	Full Scale Adjustment Offset
AnalogOutZeroPreset	/ao**n**04	D	Zero Adjustment Preset Value
AnalogOutSpanPreset	/ao**n**04	D	Full Scale Adjustment Present Value

Ladder Logic Data

These variables hold the values associated with ladder logic shared data. The fields are **external read only**.

Local Method	Local File Id	Internal Format	External Format
LadderRungCounter	/lad01	US	Number 'n' of rungs in ladder
LadderRungs	/lad02	600C	Ladder, containing 'n' rungs

BASIC Application Shared Data

These variables hold the values associated with BASIC application shared data. The fields are **external read/write**.

Local Method	Local File ID	Internal Format	External Format
Program 1	/bas01	20C	19 Alphanumeric characters + 0
Program 2	/bas02	20C	19 Alphanumeric characters + 0
Program 3	/bas03	20C	19 Alphanumeric characters + 0
Program 4	/bas04	20C	19 Alphanumeric characters + 0
Program 5	/bas05	20C	19 Alphanumeric characters + 0
Program 6	/bas06	20C	19 Alphanumeric characters + 0
Program 7	/bas07	20C	19 Alphanumeric characters + 0
Program 8	/bas08	20C	19 Alphanumeric characters + 0
Program 9	/bas09	20C	19 Alphanumeric characters + 0
KeyboardSource	/bas10	UC	0=None, 1=Keypad, 2=Keyboard, 3=Both
DisplayDestination	/bas11	UC	0=None, 1=Lower Display, 2=Serial Port

Local Method	Local File ID	Internal Format	External Format
ProgrammableTareWeightScaleA	/bas12	D	double float weight
ProgrammableTareWeightScaleB	/bas13	D	double float weight
JagBASIC applications use these fields to communicate custom fields with a PLC. Scale A and Scale B have unique shared data field names. The floating point and string fields are each four bytes long. The PLC and the JagBASIC application define the meaning of the fields. The terminal sends the PLC input fields designated as "Real-Time" to the PLC at every weight update. It sends or receives the other fields only when the PLC specifically requests them. You can also use these shared data variables as sources for Analog Output channel 1, channel 2, or both channels. The JagBASIC source variable for channel 1 is floating point variable /bas18. The JagBASIC source variable for channel 2 is floating point variable /bas20. You can use a JagBASIC source for one channel and scale source for the other channel.			
CustomOutput_A1_FromPLC	/bas14	F	Float. Defined by user. Scale A. Custom Output 1 to Scale A from PLC.
CustomOutput_A2_FromPLC	/bas15	4C	String. Defined by user. Scale A. Custom Output 2 to Scale A from PLC.
CustomOutput_A3_FromPLC	/bas16	F	Float. Defined by user. Scale A. Custom Output 3 to Scale A from PLC.
CustomOutput_A4_FromPLC	/bas17	4C	String. Defined by user. Scale A. Custom Output 4 to Scale A from PLC.
CustomInput_A1_ToPLC	/bas18	F	Float. Defined by user. Scale A. Real-Time. Custom Input 1 from Scale A to PLC.
CustomInput_A2_ToPLC	/bas19	4C	String. Defined by user. Scale A. Real-Time. Custom Input 2 from Scale A to PLC.
CustomInput_A3_ToPLC	/bas20	F	Float. Defined by user. Scale A. Custom Input 3 from Scale A to PLC.
CustomInput_A4_ToPLC	/bas21	4C	String. Defined by user. Scale A. Custom Input 4 from Scale A to PLC.
CustomOutput_B1_FromPLC	/bas22	F	Float. Defined by user. Scale B. Custom Output 1 to Scale B from PLC.
CustomOutput_B2_FromPLC	/bas23	4C	String. Defined by user. Scale B. Custom Output 2 to Scale B from PLC.
CustomOutput_B3_FromPLC	/bas24	F	Float. Defined by user. Scale B. Custom Output 3 to Scale B from PLC.
CustomOutput_B4_FromPLC	/bas25	4C	String. Defined by user. Scale B. Custom Output 4 to Scale B from PLC.
CustomInput_B1_ToPLC	/bas26	F	Float. Defined by user. Scale B. Real-Time. Custom Input 1 from Scale B to PLC.
CustomInput_B2_ToPLC	/bas27	4C	String. Defined by user. Scale B. Real-Time. Custom Input 2 from Scale B to PLC.
CustomInput_B3_ToPLC	/bas28	F	Float. Defined by user. Scale B. Custom Input 3 from Scale B to PLC.
CustomInput_B4_ToPLC	/bas29	4C	String. Defined by user. Scale B. Custom Input 4 from Scale B to PLC.
CustomOutput_C1_FromPLC	/bas30	F	Float. Defined by user. Scale C.
CustomOutput_C2_FromPLC	/bas31	4C	String. Defined by user. Scale C.
CustomOutput_C3_FromPLC	/bas32	F	Float. Defined by user. Scale C.
CustomOutput_C4_FromPLC	/bas33	4C	String. Defined by user. Scale C.

Local Method	Local File ID	Internal Format	External Format
CustomInput_C1_ToPLC	/bas34	F	Float. Defined by user. Scale C. High Speed.
CustomInput_C2_ToPLC	/bas35	4C	String. Defined by user. Scale C. High Speed.
CustomInput_C3_ToPLC	/bas36	F	Float. Defined by user. Scale C.
CustomInput_C4_ToPLC	/bas37	4C	String. Defined by user. Scale C.
CustomOutput_D1_FromPLC	/bas38	F	Float. Defined by user. Scale D.
CustomOutput_D2_FromPLC	/bas39	4C	String. Defined by user. Scale D.
CustomOutput_D3_FromPLC	/bas40	F	Float. Defined by user. Scale D.
CustomOutput_D4_FromPLC	/bas41	4C	String. Defined by user. Scale D.
CustomInput_D1_ToPLC	/bas42	F	Float. Defined by user. Scale D. High Speed.
CustomInput_D2_ToPLC	/bas42	4C	String. Defined by user. Scale D. High Speed.
CustomInput_D3_ToPLC	/bas44	F	Float. Defined by user. Scale D.
CustomInput_D4_ToPLC	/bas45	4C	String. Defined by user. Scale D.
CustomOutput_E1_FromPLC	/bas46	F	Float. Defined by user. Scale E.
CustomOutput_E2_FromPLC	/bas47	4C	String. Defined by user. Scale E.
CustomOutput_E3_FromPLC	/bas48	F	Float. Defined by user. Scale E.
CustomOutput_E4_FromPLC	/bas49	4C	String. Defined by user. Scale E.
CustomInput_E1_ToPLC	/bas50	F	Float. Defined by user. Scale E. High Speed.
CustomInput_E2_ToPLC	/bas51	4C	String. Defined by user. Scale E. High Speed.
CustomInput_E3_ToPLC	/bas52	F	Float. Defined by user. Scale E.
CustomInput_E4_ToPLC	/bas53	4C	String. Defined by user. Scale E.
ProgrammableTareWeightScaleC	/bas54	D	double float weight
ProgrammableTareWeightScaleD	/bas55	D	double float weight
ProgrammableTareWeightScaleE	/bas56	D	double float weight
AutoStartEnabled	/bas85	US bit	1=True, 0=False
EscapeEnabled	/bas86	US bit	1=True, 0=False
SelectEnabled	/bas87	US bit	1=True, 0=False
ManualStartEnabled	/bas88	US bit	1=True, 0=False
ManualStopEnabled	/bas89	US bit	1=True, 0=False

Power Cell Log

The fields are **external read only**.

Local Method	Local File ID	Internal Format	External Format
NumberErrors_Cell 1-24	/pce01	24D	192 bytes. One double float entry for each cell.
Calibrated Zero Count-Cell 1-24	/pce02	24D	192 bytes. One double float entry for each cell.
Current Zero Counts_Cell 1-24	/pce03	20C	192 bytes. One double float entry for each cell.

Shared Data EEPROM Elements

This section lists the shared data EEPROM elements. These variables hold the values associated with different erasable programmable read-only memory elements.

Scale Calibration Parameters Stored in EEPROM

These shared data variables hold the values associated with scale calibration parameters stored in the EEPROM. The fields are **external read only**. The '**n**' will be replaced with the Internal Scale number. The scale number can be from 1 to 5. The Scale 5 parameters, or summing scale parameters are stored in BRAM rather than EEPROM.

Local Method	Local File ID	Internal Format	External Format
AddressOfFirstLoadCell	/ce**n**01	UC	POWERCELL starting address
NumberLoadCells	/ce**n**02	UC	unsigned 0-255
PrimaryUnits	/ce**n**03	UC	1 alphanumeric (**1=**pounds, **2=**kilograms, **3=**grams, or 4= metric tons)
PrimaryNumberRanges	/ce**n**04	UC	1 alphanumeric
PrimaryLowIncrementSize	/ce**n**05	D	double float weight
PrimaryMidIncrementSize	/ce**n**06	D	double float weight
PrimaryHighIncrementSize	/ce**n**07	D	double float weight
PrimaryLowMidThreshold	/ce**n**08	D	double float weight
PrimaryMidHighThreshold	/ce**n**09	D	double float weight
PrimaryScaleCapacity	/ce**n**10	D	double float weight
SecondaryUnits	/ce**n**11	UC	1 alphanumeric (**1=**pounds, **2=**kilograms, **3=**grams, or 4=metric tons)
SecondaryNumberRanges	/ce**n**12	UC	1 alphanumeric
SecondaryLowIncrementSize	/ce**n**13	D	double float weight
SecondaryMidIncrementSize	/ce**n**14	D	double float weight
SecondaryHighIncrementSize	/ce**n**15	D	double float weight
SecondaryLowMidThreshold	/ce**n**16	D	double float weight
SecondaryMidHighThreshold	/ce**n**17	D	double float weight
SecondaryScaleCapacity	/ce**n**18	D	double float weight
CalibrationUnits	/ce**n**19	UC	1 alphanumeric (**1=**primary or **2=**secondary)
ZeroCalibrationCounts	/ce**n**20	L	integer
HighCalibrationCounts	/ce**n**21	L	integer
HighCalibrationWeight	/ce**n**22	D	double float weight
MidCalibrationCounts	/ce**n**23	L	integer
MidCalibrationWeight	/ce**n**24	D	double float weight
GravityAdjust	/ce**n**25	D	double float
MotionStabilitySensitivityinD	/ce**n**26	F	float divisions
MotionStabilityTimePeriod	/ce**n**27	UC	(1=3 sec, ..., 7=10sec)
ScaleSerialNumber	/ce**n**28	12C	12 alphanumeric

CalibrationCounter1	/ce**n**29	UC	1 byte unsigned binary
CalibrationCounter2	/ce**n**30	UC	1 byte unsigned binary
AtoD Update Rate	/ce**n**31	UC	Conversions / Second (1-255)
OverCapacityDivisions	/ce**n**32	UC	number of divisions (1-255)
LinearityCorrectionEnable	/ce**n**85	US bit	1=True, 0=False
OverCapacityBlanking	/ce**n**86	US bit	1=True, 0=False
MultirangeMode	/ce**n**87	US bit	1=Auto, 0=Manual
Shift Adjust Mode	/ce**n**88	US bit	0=Cell, 1=Pair

EEPROM Version Identification

These shared data variables hold the values associated with EEPROM version identification. The fields are **external read only**.

Local Method	Local File ID	Internal Format	External Format
EEPROMVersionId (Scale A)	/ee101	L	Long Integer (32 bits)
EEPROMVersionId (Scale B)	/ee201	L	Long Integer (32 bits)

Shift Adjust Variables

These variables hold the values associated with shift adjust shared data. The fields are **external read only**. The '**n**' will be replaced with the Internal Scale number.

Local Method	Local File ID	Internal Format	External Format
Cell #1 Shift Constants	/sa**n**01	L	Long Integer (32 bits) Normalized
Cell #16 Shift Constants	/sa**n**16	L	Long Integer (32 bits) Normalized

Expanded Shift Adjust Variables

These variables hold the values associated with expanded shift adjust shared data. The fields are external read only. The '**n**' will be replaced with the Internal Scale number.

Local Method	Local File ID	Internal For mat	External Format
Cell #17 Shift Constants	/sx**n**17	L	Long Integer (32 bits) Normalized
Cell #24 Shift Constants	/sx**n**24	L	Long Integer (32 bits) Normalized

Scale Zero Shared Data

These variables hold the values associated with scale zero shared data. The fields are external read only. The 'n' will be replaced with the Internal Scale number.

Local Method	Local File ID	Internal Format	External Format
PowerUpZeroCapturePosRange	/zr**n**01	UC	percent capacity (0-99)
PowerUpZeroCaptureNegRange	/zr**n**02	UC	percent capacity (0-99)
PushbuttonZeroPosRange	/zr**n**03	UC	percent capacity (0-99)
PushbuttonZeroNegRange	/zr**n**04	UC	percent capacity (0-99)
AutoZeroMaintWindow	/zr**n**05	F	floatnumber of divisions
BehindZeroDivisions	/zrn06	UC	0-99 divisions
PushbuttonZero	/zr**n**85	US bit	1=True, 0=False
AutoZeroGross	/zr**n**86	US bit	1=True, 0=False
AutoZeroGross_Net	/zr**n**87	US bit	1=True, 0=False
ZeroIndicationGross	/zr**n**88	US bit	1=True, 0=False
ZeroIndicationGross_Net	/zr**n**89	US bit	1=True, 0=False

NOTES

7

Global Discrete I/O Data

Global Discrete I/O data has bit fields representing physical discrete I/O and logical I/O bits. The logical I/O may be either level-sensitive states or edge- sensitive events. Global Discrete I/O is transitory data in that it is not saved during a power-down. It is initialized to zero and then regenerated on power-up. These bit fields are the "contacts" and "coils" for the ladder logic processor.

Level-Sensitive, Logical Discrete I/O Data

Level-sensitive fields can generate callbacks when either a 0 or a 1 is written to the field. Field names starting with **s_2** are input contacts to the ladder processor. Field names starting with **s_6** are coils for the ladder processor.

For all level-sensitive logical I/O data the following apply:

Internal Format	External Format	Condition
G	US Bit	1 = True, 0 = False

Description	Local Field	Comments
The terminal O/S sets the following fields to reflect the status of Scale A and ScaleB.		
MotionOut_A	/s_200	Read only
CenterOfZero_A	/s_201	Read only
OverCapacity_A	/s_202	Read only
UnderZero_A	/s_203	Read only
NetMode_A	/s_204	Read only
ScaleCriticalError_A	/s_205	Read only
StoredWeightMode_A	/s_206	Read only
ScaleSelected_A	/s_207	Read only
IDNET_In_Motion_Error_A	/s_260	Reserved for terminal O/S use only.
WeightDataOK_A	/s_261	Read only
RateSetpointOK_A	/s_262	Read only
The O/S sets the following fields to report on the status of Scale B		
MotionOut_B	/s_208	Read only
CenterOfZero_B	/s_209	Read only
OverCapacity_B	/s_20a	Read only
UnderZero_B	/s_20b	Read only
NetMode_B	/s_20c	Read only
ScaleCriticalError_B	/s_20d	Read only
StoredWeightMode_B	/s_20e	Read only
ScaleSelected_B	/s_20f	Read only
IDNET_In_Motion_Error_B	/s_268	Reserved for terminal O/S use only.
WeightDataOK_B	/s_269	Read only
RateSetpointOK_B	/s_26a	Read only
The terminal O/S sets the following fields to report status of Scale C.		

Description	Local Field	Comments
MotionOut_C	/s_270	Read only
CenterOfZero_C	/s_271	Read only
OverCapacity_C	/s_272	Read only
UnderZero_C	/s_273	Read only
NetMode_C	/s_274	Read only
ScaleCriticalError_C	/s_275	Read only
StoredWeightMode_C	/s_276	Read only
ScaleSelected_C	/s_277	Read only
IDNET_In_Motion_Error_C	/s_278	Reserved for terminal O/S use only.
WeightDataOK_C	/s_279	Read only
RateSetpointOK_C	/s_27a	Read only
The terminal O/S sets the following fields to report status of Scale D.		
MotionOut_D	/s_280	Read only
CenterOfZero_D	/s_281	Read only
OverCapacity_D	/s_282	Read only
UnderZero_D	/s_283	Read only
NetMode_D	/s_284	Read only
ScaleCriticalError_D	/s_285	Read only
StoredWeightMode_D	/s_286	Read only
ScaleSelected_D	/s_287	Read only
IDNET_In_Motion_Error_D	/s_288	Reserved for terminal O/S use only.
WeightDataOK_D	/s_289	Read only
RateSetpointOK_D	/s_28a	Read only
The Terminal O/S sets the following fields to report status of Scale E. (Summing Scale)		
MotionOut_E	/s_2f0	Read only
CenterOfZero_E	/s_2f1	Read only
OverCapacity_E	/s_2f2	Read only
UnderZero_E	/s_2f3	Read only
NetMode_E	/s_2f4	Read only
ScaleCriticalError_E	/s_2f5	Read only
StoredWeightMode_E	/s_2f6	Read only
ScaleSelected_E	/s_2f7	Read only
IDNET_In_Motion_Error_E	/s_2f8	Reserved for terminal O/S use only.
WeightDataOK_E	/s_2f9	Read only
RateSetpointOK_E	/s_2fa	Read only
Terminal O/S sets the following fields to reflect the status of Setpoints 1-12.		
SetpointFeeding_1	/s_210	Read only
SetpointFastFeeding_1	/s_211	Read only
SetpointWithinTolerance_1	/s_212	Read only
SetpointFeeding_2	/s_214	Read only
SetpointFastFeeding_2	/s_215	Read only
SetpointWithinTolerance_2	/s_216	Read only
SetpointFeeding_3	/s_218	Read only
SetpointFastFeeding_3	/s_219	Read only
SetpointWithinTolerance_3	/s_21a	Read only
SetpointFeeding_4	/s_21c	Read only
SetpointFastFeeding_4	/s_21d	Read only

Description	Local Field	Comments
SetpointWithinTolerance_4	/s_21e	Read only
SetpointFeeding_5	/s_220	Read only
SetpointFastFeeding_5	/s_221	Read only
SetpointWithinTolerance_5	/s_222	Read only
SetpointFeeding_6	/s_224	Read only
SetpointFastFeeding_6	/s_225	Read only
SetpointWithinTolerance_6	/s_226	Read only
SetpointFeeding_7	/s_228	Read only
SetpointFastFeeding_7	/s_229	Read only
SetpointWithinTolerance_7	/s_22a	Read only
SetpointFeeding_8	/s_22c	Read only
SetpointFastFeeding_8	/s_22d	Read only
SetpointWithinTolerance_8	/s_22e	Read only
SetpointFeeding_9	/s_230	Read only
SetpointFastFeeding_9	/s_231	Read only
SetpointWithinTolerance_9	/s_232	Read only
SetpointFeeding_10	/s_234	Read only
SetpointFastFeeding_10	/s_235	Read only
SetpointWithinTolerance_10	/s_236	Read only
SetpointFeeding_11	/s_238	Default is ZERO TOL A. Read only
SetpointFastFeeding_11	/s_239	Read only
SetpointWithinTolerance_11	/s_23a	Read only
SetpointFeeding_12	/s_23c	Default is ZERO TOL B. Read only
SetpointFastFeeding_12	/s_23d	Read only
SetpointWithinTolerance_12	/s_23e	Read only
Terminal O/S sets the following fields to give the status of the Ethernet connections.		
NodeOnLine_1	/s_241	Read only
NodeOnLine_2	/s_242	Read only
NodeOnLine_3	/s_243	Read only
NodeOnLine_4	/s_244	Read only
NodeOnLine_5	/s_245	Read only
NodeOnLine_6	/s_246	Read only
HostOnLine_3	/s_24d	Read only
HostOnLine_2	/s_24e	Read only
HostOnLine_1	/s_24f	Read only

Description	Local Field	Comments
JagBASIC applications can set the following four discrete bit fields to send real-time status data to a PLC.		
PLC_CustomStatus1_Scale_A	/s_250	Read/Write. Custom Real-Time Status 1 from Scale A to PLC.
PLC_CustomStatus2_Scale_A	/s_251	Read/Write. Custom Real-Time Status 2 from Scale A to PLC.
PLC_CustomStatus1_Scale_B	/s_252	Read/Write. Custom Real-Time Status 1 from Scale B to PLC.
PLC_CustomStatus2_Scale_B	/s_253	Read/Write. Custom Real-Time Status 2 from Scale B to PLC.
PLC_CustomStatus1_Scale_C	/s_254	Read/Write. Custom Real-Time Status 1 from Scale C to PLC.
PLC_CustomStatus2_Scale_C	/s_255	Read/Write. Custom Real-Time Status 2 from Scale C to PLC.
PLC_CustomStatus1_Scale_D	/s_256	Read/Write. Custom Real-Time Status 1 from Scale D to PLC.
PLC_CustomStatus2_Scale_D	/s_257	Read/Write. Custom Real-Time Status 2 from Scale D to PLC.
PLC_CustomStatus1_Scale_E	/s_258	Read/Write. Custom Real-Time Status 1 from Scale E to PLC.
PLC_CustomStatus2_Scale_E	/s_259	Read/Write. Custom Real-Time Status 2 from Scale E to PLC.
Terminal O/S sets the following fields to report success (=0) or error (=1) when an external agency uses a corresponding discrete field to trigger a command in the terminal O/S.		
Command Staus Bits for Scale A		
TareScaleError_A	/s_290	Read only
ClearTareScaleError_A	/s_291	Read only
PrintScaleError_A	/s_292	Read only
ZeroScaleError_A	/s_293	Read only
SwitchToPrimUnitsError_A	/s_294	Read only
SwitchToSecondUnitsError_A	/s_295	Read only
SwitchToOtherUnitsError_A	/s_296	Read only
ApplySetupError_A	/s_297	Read only

Description	Local Field	Comments
RestartSetpointsError_A	/s_298	Read only
RestartRateCalculationError_A	/s_299	Read only
RestartFilterError_A	/s_29a	Read only
RestartSetpointCoincidenceError_A	/s_29b	Read only
DisableScaleError_A	/s_29c	Read only
CapturePowerCellCountError_A	/t_29d	Read Only
WriteCal.ToEEPromErrorA	/t_29e	Read Only
Command Staus Bits for Scale B		
TareScaleError_B	/s_2a0	Read only
ClearTareScaleError_B	/s_2a1	Read only
PrintScaleError_B	/s_2a2	Read only
ZeroScaleError_B	/s_2a3	Read only
SwitchToPrimUnitsError_B	/s_2a4	Read only
SwitchToSecondUnitsError_B	/s_2a5	Read only
SwitchToOtherUnitsError_B	/s_2a6	Read only
ApplySetupError_B	/s_2a7	Read only
RestartSetpointsError_B	/s_2a8	Read only
RestartRateCalculationError_B	/s_2a9	Read only
RestartFilterError_B	/s_2aa	Read only
RestartSetpointCoincidenceError_B	/s_2ab	Read only
DisableScaleError_B	/s_2ac	Read only
CapturePowerCellCountError_B	/t_2ad	Read Only
WriteCal.ToEEPromErrorB	/t_2ae	Read Only
Command Staus Bits for Selected Scale		
TareScaleError_SelectedScale	/s_2b0	Read only
ClearTareScaleError_SelectedScale	/s_2b1	Read only
PrintScaleError_SelectedScale	/s_2b2	Read only
ZeroScaleError_SelectedScale	/s_2b3	Read only
SwitchToPrimUnitsError_SelScl	/s_2b4	Read only
SwitchToSecondUnitsError_SelScl	/s_2b5	Read only
SwitchToOtherUnitsError_SelScl	/s_2b6	Read only
Command Staus Bits for Custom Print		
CustomPrintError_1	/s_2b7	Read only
CustomPrintError_2	/s_2b8	Read only
CustomPrintError_3	/s_2b9	Read only
CustomPrintError_4	/s_2ba	Read only
CustomPrintError_5	/s_2bb	Read only
Command Status Bits for Scale C		
JagBasicEnabled	/2_sbf	Read only
TareScaleError_C	/s_2c0	Read only
ClearTareScaleError_C	/s_2c1	Read only
PrintScaleError_C	/s_2c2	Read only
ZeroScaleError_C	/s_2c3	Read only

Description	Local Field	Comments
SwitchToPrimUnitsError_C	/s_2c4	Read only
SwitchToSecondUnitsError_C	/s_2c5	Read only
SwitchToOtherUnitsError_C	/s_2c6	Read only
ApplySetupError_C	/s_2c7	Read only
RestartSetpointsError_C	/s_2c8	Read only
RestartRateCalculationError_C	/s_2c9	Read only
RestartFilterError_C	/s_2ca	Read only
RestartSetpointCoincidenceError_C	/s_2cb	Read only
DisableScaleError_C	/s_2cc	Read only
CapturePowerCellCountError_C	/t_2cd	Read Only
WriteCal.ToEEPromErrorC	/t_2ce	Read Only
Command Status Bits for Scale D		
TareScaleError_D	/s_2d0	Read only
ClearTareScaleError_D	/s_2d1	Read only
PrintScaleError_D	/s_2d2	Read only
ZeroScaleError_D	/s_2d3	Read only
SwitchToPrimUnitsError_D	/s_2d4	Read only
SwitchToSecondUnitsError_D	/s_2d5	Read only
SwitchToOtherUnitsError_D	/s_2d6	Read only
ApplySetupError_D	/s_2d7	Read only
RestartSetpointsError_D	/s_2d8	Read only
RestartRateCalculationError_D	/s_2d9	Read only
RestartFilterError_D	/s_2da	Read only
RestartSetpointCoincidenceError_D	/s_2db	Read only
DisableScaleError_D	/s_2dc	Read only
CapturePowerCellCountError_D	/t_2dd	Read Only
WriteCal.ToEEPromErrorD	/t_2de	Read Only
Command Status Bits for Scale E		
TareScaleError_E	/s_2e0	Read only
ClearTareScaleError_E	/s_2e1	Read only
PrintScaleError_E	/s_2e2	Read only
ZeroScaleError_E	/s_2e3	Read only
SwitchToPrimUnitsError_E	/s_2e4	Read only
SwitchToSecondUnitsError_E	/s_2e5	Read only
SwitchToOtherUnitsError_E	/s_2e6	Read only
ApplySetupError_E	/s_2e7	Read only
RestartSetpointsError_E	/s_2e8	Read only
RestartRateCalculationError_E	/s_2e9	Read only
RestartFilterError_E	/s_2ea	Read only
RestartSetpointCoincidenceError_E	/s_2eb	Read only
DisableScaleError_E	/s_2ec	Read only
CapturePowerCellCountError_E	/t_2ed	Read Only
WriteCal.ToEEPromErrorE	/t_2ee	Read Only

Miscellaneous Status Bits		
SelectScaleError_A	/s_2c0	Read only Rel. M
SelectScaleError_B	/s_2c1	Read only Rel. M
SelectOtherScaleError	/s_2c2	Read only Rel. M
DemandCustomPrintError_1	/s_2c3	Read only Rel. M
DemandCustomPrintError_2	/s_2c4	Read only Rel. M
DemandCustomPrintError_3	/s_2c5	Read only Rel. M
DemandCustomPrintError_4	/s_2c6	Read only Rel. M
DemandCustomPrintError_5	/s_2c7	Read only Rel. M
JagBASICEnabled	/s_2d0	Read only Rel. M
Miscellaneous Triggers		
MasterControlRelay	/s_600	Shuts down all I/O. Read/Write
DisableErrorDisplay	/s_603	Read/Write
DisableNumericDisplay	/s_604	Read/Write
Disable Setup	/s_609	Read/Write
Disable Keypag	/s_60a	Read/Write
Disable Qwerty PG keys postioning, (home, end, etc…)	/s_60b	Read/Write

Edge-Sensitive, Logical Discrete I/O Data

Edge-sensitive bit fields only trigger events when a 1 is written to the field. They are ladder logic coils. If an error occurs in the event, the task writes a 1 into the corresponding error bit. If the event is successful, it writes a 0 on completion.

For all edge-sensitive logical discrete I/O data the following apply:

Internal Format	External Format	Condition
G	US bit	1 = Trigger, 0 = Complete

Fields are external read/write.

Description	Local Field
Terminal O/S sets the following fields to indicate when the terminal has calculated a new weight value. A JagBASIC application can use events to monitor these fields. It must set the field to 0 before the same event will trigger again.	
WeightUpdated_A	/t_688
WeightUpdated_B	/t_689
WeightUpdated_C	/t_613
WeightUpdated_D	/t_614
WeightUpdated_E	/t_615

Description	Local Field
Terminal O/S sets these discrete fields =1 whenever it installs a new setpoint. A JagBASIC application can use events to monitor these fields. It must set the field to 0 before the same event will trigger again.	
SetpointInstalled_A	/t_68c
SetpointInstalled_B	/t_68d
SetpointInstalled_C	/t_616
SetpointInstalled_D	/t_617
SetpointInstalled_E	/t_618
CalibrationComplete_A	/t_68e
CalibrationComplete_B	/t_68f
CalibrationComplete_C	/t_619
CalibrationComplete_D	/t_61a
CalibrationComplete_E	/t_61b
External agencies can set the following fields to trigger a command within the terminal O/S. The terminal O/S sets the field to 0 when it is done processing the command. It will also set a corresponding error bit to indicate when there is an error in processing the command.	
Triggers for Scale A	
TareScale_A	/t_690
ClearTareScale_A	/t_691
PrintScale_A	/t_692
ZeroScale_A	/t_693
SwitchToPrimaryUnits_A	/t_694
SwitchToSecondUnits_A	/t_695
SwitchToOtherUnits_A	/t_696
ApplySetup_A	/t_697
RestartSetpoints_A	/t_698
RestartRateCalculation_A	/t_699
RestartFilter_A	/t_69a
ResetSetpointCoincidence_A	/t_69b
DisableScale_A	/t_69c
CapturePowerCellCounts_A	/t_69d
WriteCalibrationToEEProm_A	/t_69e
Triggers for Scale B	
TareScale_B	/t_6a0
ClearTareScale_B	/t_6a1
PrintScale_B	/t_6a2
ZeroScale_B	/t_6a3
SwitchToPrimaryUnits_B	/t_6a4
SwitchToSecondUnits_B	/t_6a5
SwitchToOtherUnits_B	/t_6a6
ApplySetup_B	/t_6a7
RestartSetpoints_B	/t_6a8
RestartRateCalculation_B	/t_6a9
RestartFilter_B	/t_6aa

Description	Local Field
ResetSetpointCoincidence_B	/t_6ab
DisableScale_B	/t_6ac
CapturePowerCellCounts_B	/t_6ad
WriteCalibrationToEEProm_B	/t_6ae
Triggers for Scale C	
TareScale_C	/t_620
ClearTareScale_ C	/t_621
PrintScale_C	/t_622
ZeroScale_C	/t_623
SwitchToPrimaryUnits_C	/t_624
SwitchToSecondUnits_C	/t_625
SwitchToOtherUnits_C	/t_626
ApplySetup_C	/t_627
RestartSetpoints_C	/t_628
RestartRateCalculation_C	/t_629
RestartFilter_C	/t_62a
ResetSetpointCoincidence_C	/t_62b
DisableScale_C	/t_62c
CapturePowerCellCounts_C	/t_62d
WriteCalibrationToEEProm_C	/t_62e
Triggers for Scale D	
TareScale_D	/t_630
ClearTareScale_D	/t_631
PrintScale_D	/t_632
ZeroScale_D	/t_633
SwitchToPrimaryUnits_D	/t_634
SwitchToSecondUnits_D	/t_635
SwitchToOtherUnits_D	/t_636
ApplySetup_D	/t_637
RestartSetpoints_D	/t_638
RestartRateCalculation_D	/t_639
RestartFilter_D	/t_63a
ResetSetpointCoincidence_D	/t_63b
DisableScale_D	/t_63c
CapturePowerCellCounts_D	/t_63d
WriteCalibrationToEEProm_D	/t_63e
Triggers for Scale E	
TareScale_E	/t_640
ClearTareScale_E	/t_641
PrintScale_E	/t_642
ZeroScale_E	/t_643
SwitchToPrimaryUnits_E	/t_644
SwitchToSecondUnits_E	/t_645
SwitchToOtherUnits_E	/t_646

Description	Local Field
ApplySetup_E	/t_647
RestartSetpoints_E	/t_648
RestartRateCalculation_E	/t_649
RestartFilter_E	/t_64a
ResetSetpointCoincidence_E	/t_64b
DisableScale_E	/t_64c
CapturePowerCellCounts_E	/t_64d
WriteCalibrationToEEProm_E	/t_64e
Triggers for Selected Scale	
TareScale_SelectedScale	/t_6b0
ClearTareScale_SelectedScale	/t_6b1
PrintScale_SelectedScale	/t_6b2
ZeroScale_SelectedScale	/t_6b3
SwitchToPrimaryUnits_SelScl	/t_6b4
SwitchToSecondUnits_SelScl	/t_6b5
SwitchToOtherUnits_SelScl	/t_6b6
SelectScale_A	/t_6c0
SelectScale_B	/t_6c1
SelectScale_C	/t_650
SelectScale_D	/t_651
SelectScale_E	/t_652
SelectOtherScale	/t_6c2
Custom Print Triggers	
DemandCustomPrint_1*	/t_6c3
DemandCustomPrint_2*	/t_6c4
DemandCustomPrint_3*	/t_6c5
DemandCustomPrint_4*	/t_6c6
DemandCustomPrint_5*	/t_6c7
A PC Host sets the following four discrete bit fields to send real-time commands to a JagBASIC application.	
CustomCommand1	/t_6cc
CustomCommand2	/t_6cd
CustomCommand3	/t_6ce
CustomCommand4	/t_6cf
Terminal O/S sets these fields =1 whenever it detects a rising or falling edge in the discrete inputs. A JagBASIC application can use events to monitor these fields. It must set the field to 0 before the same event will trigger again.	
DiscreteInputRisingEdge_1	/p_6e0
DiscreteInputRisingEdge_2	/p_6e1
DiscreteInputRisingEdge_3	/p_6e2
DiscreteInputRisingEdge_4	/p_6e3
DiscreteInputRisingEdge_5	/p_6e8
DiscreteInputRisingEdge_6	/p_6e9
DiscreteInputRisingEdge_7	/p_6ea
DiscreteInputRisingEdge_8	/p_6eb

*Enable Custom Print in Serial Setup to enable JagBASIC to print using Demand Custom Print.

Description	Local Field
DiscreteInputRisingEdge_9	/p_6ec
DiscreteInputRisingEdge_10	/p_6ed
DiscreteInputRisingEdge_11	/p_6ee
DiscreteInputRisingEdge_12	/p_6ef
DiscreteInputFallingEdge_1	/p_6f0
DiscreteInputFallingEdge_2	/p_6f1
DiscreteInputFallingEdge_3	/p_6f2
DiscreteInputFallingEdge_4	/p_6f3
DiscreteInputFallingEdge_5	/p_6f8
DiscreteInputFallingEdge_6	/p_6f9
DiscreteInputFallingEdge_7	/p_6fa
DiscreteInputFallingEdge_8	/p_6fb
DiscreteInputFallingEdge_9	/p_6fc
DiscreteInputFallingEdge_10	/p_6fd
DiscreteInputFallingEdge_11	/p_6fe
DiscreteInputFallingEdge_12	/p_6ff

Physical Discrete I/O Data

Physical discrete input and output data is stored on the Controller and Multi-Function Boards. The stored logical 1s or 0s correspond to whether a physical discrete input or output is true or false and on or off.

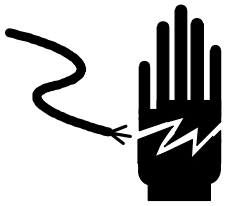
For all physical discrete I/O data the following apply:

Internal Format	External Format	Condition
G	US bit	1 = Trigger, 0 = Complete

Description	Local Field	Comment
PhysicalDiscreteInput_1	/p_100	Read only
PhysicalDiscreteInput_2	/p_101	Read only
PhysicalDiscreteInput_3	/p_102	Read only
PhysicalDiscreteInput_4	/p_103	Read only
PhysicalDiscreteInput_5	/p_108	Read only
PhysicalDiscreteInput_6	/p_109	Read only
PhysicalDiscreteInput_7	/p_10a	Read only
PhysicalDiscreteInput_8	/p_10b	Read only
PhysicalDiscreteInput_9	/p_10c	Read only
PhysicalDiscreteInput_10	/p_10d	Read only
PhysicalDiscreteInput_11	/p_10e	Read only
PhysicalDiscreteInput_12	/p_10f	Read only
PhysicalDiscreteOutput_1	/p_500	Read/Write
PhysicalDiscreteOutput_2	/p_501	Read/Write

Description	Local Field	Comment
PhysicalDiscreteOutput_3	/p_502	Read/Write
PhysicalDiscreteOutput_4	/p_503	Read/Write
PhysicalDiscreteOutput_5	/p_508	Read/Write
PhysicalDiscreteOutput_6	/p_509	Read/Write
PhysicalDiscreteOutput_7	/p_50a	Read/Write
PhysicalDiscreteOutput_8	/p_50b	Read/Write
PhysicalDiscreteOutput_9	/p_50c	Read/Write
PhysicalDiscreteOutput_10	/p_50d	Read/Write
PhysicalDiscreteOutput_11	/p_50e	Read/Write
PhysicalDiscreteOutput_12	/p_50f	Read/Write

8 Sample Application Programs



WARNING

PERMIT ONLY QUALIFIED PERSONNEL TO SERVICE THIS EQUIPMENT. EXERCISE CARE WHEN MAKING CHECKS, TESTS AND ADJUSTMENTS THAT MUST BE MADE WITH POWER ON. FAILING TO OBSERVE THESE PRECAUTIONS CAN RESULT IN BODILY HARM OR PROPERTY DAMAGE.

CAUTION

THESE PROGRAMS ARE ONLY INTENDED TO DEMONSTRATE THE PROGRAMMING FLEXIBILITY OF JAGBASIC. THEY MAY NOT APPLY TO YOUR SPECIFIC APPLICATION! ONLY PERMIT QUALIFIED PERSONNEL TO CREATE JAGBASIC PROGRAMS.

This section contains examples of application programs that can be used as starting points in creating your own JagBASIC programs. They include examples which:

- Display the weight of Scale A.
- Display/toggle Scale A and Scale B.
- Clear random access files.
- Generate continuous output.
- Display the setpoint value.
- Weigh inbound/outbound trucks.
- Perform manual batching.
- Count parts.

Display Scale A Weight

Only a few lines of code are required to create a JagBASIC program. For example, this short program displays Scale A on the lower terminal display.

```
10 DEFSHR gross$,wt101
20 PRINT " W =";gross$
30 GOTO 20
40 END
```

This program can be created in one of the following ways:

- Typed in on the terminal.
- Created in a text editor on a PC and downloaded to the terminal using the program download command SZ.
- Created in a text editor on a PC and downloaded with a communication program supporting Zmodem.

You can then perform the following operations on the program:

- To execute the program, assuming the program was typed in on the JAGXTREME terminal, type RUN at the BASIC: prompt. The weight from Scale A should display on the lower JAGXTREME terminal display.
- To end the program, press the ESC key on either the keyboard or keypad.
- To save the program, type: save "filex.bas", where x is a number from 1 to 9. For this example, we will save the program as "file1.bas".
- To call up the program file, type load "file1.bas".To run this program (or any program named file1.bas) automatically on power-up, set Autostart to Yes in the JagBASIC setup. When this feature is set to Yes, each time the terminal is powered up, the file named file1.bas (if resident in the RAMDISK) will be automatically loaded and run.
- To manually load and run finished programs, set the Manual Start feature to Yes in the JagBASIC setup. When this feature is enabled, pressing the FUNCTION key on the JAGXTREME keypad displays the prompt: [**Run Program #?**] To execute the desired program, press key 1 for file1.bas, press key 2 for file2.bas, press key 3 for file3.bas, and so on up to key 9 for file9.bas.

Display/Toggle Scale A and Scale B

This example displays and toggles the weight from Scale A or B on the upper and lower JAGXTREME displays. The programming keyboard is used to toggle the weight display.

When **A** is pressed, the upper display shows the weight from Scale A and the lower display shows the weight from Scale B.

When **B** is pressed, the upper display shows the weight from Scale B and the lower shows Scale A.

The weight is obtained from pulling the rightmost 8 characters from the standard 12 character strings wt101 and wt102.

```
10 DEFSHR w1$,wt101
20 DEFSHR w2$,wt201
30 DEFSHR sa,t_6c0
40 DEFSHR sb,t_6c1
50 sa=1
60 PRINT "Scale B=";RIGHT$(w2$,8)
70 IF INKEY$ = "b" THEN GOTO 100
80 GOTO 60
100 sb=1
110 PRINT "Scale A=";RIGHT$(w1$,8)
120 IF INKEY$ = "a" THEN GOTO 50
130 GOTO 110
140 END
```

Random Access Files

The following code segment clears a random access file to standard default ID values. Note that the JagBASIC code must re-initialize the entire output record before each "PUT" command.

```
10 OPEN "IDFILE" FOR RANDOM AS #1 LEN = 19
15 REM added line feed, carriage return for
16 REM printing out file with standard editors,
20 FIELD #1,9 AS FID$, 8 AS FWEIGHT$, 2 AS LFCR$
30 FOR X% = 1 TO 10
35  REM re-initialize record image before each "PUT"
40  LSET FID$ = "000000000" : LSET FWEIGHT$ = "00000000"
50  LSET LFCR$=CHR$(13)+CHR$(10)
60  PUT #1, X%
70 NEXT X%
80 CLOSE #1
```

The following code segment sequentially searches the random access file for an empty record in which it writes a new ID-WEIGHT record.

```
210 OPEN "IDFILE" FOR RANDOM AS #1 LEN = 19
220 FIELD #1, 9 AS FID$, 8 AS FWEIGHT$, 2 AS LFCR$
230 USEREC%=0
240 FOR REC% = 1 TO 10
250  GET #1, REC%
270  IF FID$ = "000000000" THEN USEREC% = REC% : REC%=10
280  IF EOF(1) = 1 THEN REC% = 10
290 NEXT REC%
300 LSET FWEIGHT$ = "12345.6"
310 LSET FID$="JOE TRUCK"
320 LSET LFCR$=CHR$(13)+CHR$(10)
330 IF USEREC%<>0 THEN PUT #1, USEREC%
340 CLOSE #1
```

Continuous Output

This JagBASIC program generates the standard METTLER TOLEDO continuous output for the currently selected scale, either Scale A or Scale B.

```
15 REM Preformatted Status Word A:
20 DEFSHR sw1a,wt116
30 DEFSHR sw2a,wt216
35 REM Weight units "lb", "kg" or " g":
40 DEFSHR unitA$,wt103
50 DEFSHR unitB$,wt203
55 REM Motion status
60 DEFSHR motionA,s_200
70 DEFSHR motionB,s_208
75 REM Net mode (1 = net, 0 = gross):
80 DEFSHR netA,s_204
90 DEFSHR netB,s_20c
95 REM Overcapacity status (1 = Overcapacity, 0 = not Overcapacity):
100 DEFSHR overA,s_202
110 DEFSHR overB,s_20a
115 REM Under zero status (1 = Underzero, 0 = not Underzero):
120 DEFSHR underA,s_203
```

```
130 DEFSHR underB,s_20b
135 REM Displayable net weight (with embedded decimal point)
140 DEFSHR netwtA$,wt102
150 DEFSHR netwtB$,wt202
155 REM Displayable tare weight (with embedded decimal point)
160 DEFSHR tarewtA$,ws102
170 DEFSHR tarewtB$,ws202
175 REM Selected scale
180 DEFSHR selectedScale,jag01
190 REM "width -1" suppresses LF/CR being appended LPRINT line
191 WIDTH -1
192 REM Define ASCII STX:
193 START$ = CHR$(02)
194 REM Define ASCII CR:
195 END$ = CHR$(13)
196 REM Check for selected scale here:
197 REM (Program loops back to here)
198 REM Clear status word B bits:
200 b% = 0
210 IFselectedScale = "LB" THEN GOTO 2000
1000 IFnetA=1 THEN b%=1
1020 IFnegA=1 THEN b%=b%+2
1030 IF overA=1 or underA = 1 THEN b%=b%+4
1040 IF motionA=1 THEN b%=b%+8
1050 IF unitA$="kg" THEN b%=b%+16
1070 statusBytes$=STRING$(1,sw1a)+CHR$(32+b%)+ CHR$(32)
1080 IPRINT start$+statusBytes$+RIGHT$(netwtA$,6)+RIGHT$(tarewtA$,6)+end$
1100 GOTO 200
2000 IF netB=1 THEN b%=1
2030 IF negB=1 THEN b%=b%+2
2040 IF overB=1 or underB=1 THEN b%=b%+4
2050 IF motionB=1 THEN b%=b%+8
2060 IF unitB$="kg" THEN b%=b%+16
3070 statusBytes$=string$(1,sw2a)+CHR$(32+b%)+CHR$(32)
3090 LPRINT start$+statusBytes$+RIGHT$(netwtB$,6)+RIGHT$(tarewtB$,6)+end$
3100 GOTO 200
9999 END
```

Setpoint Display

This JagBASIC program displays the setpoint value for the selected scale on the terminal lower display. Scale A uses Setpoint 1 and Scale B uses Setpoint 3. This program allows an operator on the factory floor to monitor the setpoint values for doing "hand-adds" where a remote PLC changes the setpoint values.

```
10 DEFSHR stopEnable%,bas89
20 stopEnable%=0
30 DEFSHR numScales%,jag15
40 REM INITIALIZE ONE SCALE
50 DEFSHR sp1#,sp105
60 DEFSHR units1%,ce103
70 DIM units$(3)
80 UNITS$(1)=" lb":units$(2)=" kg":units$(3)=" g"
90 U1$=units$(units1%)
100 IF numScales%=2 THEN GOTO 300
200 REM LOOP FOR ONE SCALE
```

```
210 SLEEP 900
220 PRINT "A ";sp1#;u1$
230 GOTO 210
300 REM INITIALIZE TWO SCALES
310 DEFSHR scaleID$,jag01
320 DEFSHR sp3#,sp305
330 DEFSHR units2%,ce203
340 U2$=units$(units2%)
400 REM LOOP FOR TWO SCALES
410 SLEEP 900
420 IF scaleID$="LB" THEN GOTO 450
430 PRINT "A ";sp1#;u1$
440 GOTO 410
450 PRINT "B ";sp3#;u2$
460 GOTO 410
```

Rate Calculation without the Rate Display

This is a sample JagBASIC program for setting up the rate without the rate display. This JagBASIC setup uses less of the JAGXTREME terminal's processing power than the standard control panel setup which always enables the rate display. The lower display is not constantly updated with new rate information, so it can be used for displaying more critical information.

```
5 DEFSHR ratedisp,cs188:DEFSHR auxdisp,cs189:ratedisp=0:auxdisp=0
10 DEFSHR auxunit,cs101:DEFSHR period,cs104
20 DEFSHR sample,cs105:DEFSHR freq,cs106:DEFSHR setup,t_697
30 auxunit=1:REM pounds
40 period="S":REM per second
50 sample=3:REM sample time
60 freq=1:REM interval every one second
70 setup=1:REM apply setup
80 END
```

Filling

This JagBASIC program is used for filling applications.

```
1  REM Example Filling Application
7  DEFSHR StopEnabled,bas89
8  DEFSHR SPfeeding,s_210
9  DEFSHR SPtolerance,s_212
12 DEFSHR DiscreteIn,p_100
13 DEFSHR TareA,t_690
14 DEFSHR TareAerr,s_290
15 DEFSHR DiscreteOut,p_503
16 DEFSHR NetWt,wt111
17 DEFSHR ClearTareA,t_691
18 DEFSHR MotionA,s_200
20 StopEnabled=0
60 PRINT "Place Container"
70 IF DiscreteIn=0 THEN GOTO 70
75 PRINT "Taring Container"
77 SLEEP 3000
90 TareA=1
```

```
100 IF TareA=1 THEN GOTO 100
120 IF TareAerr=0 THEN GOTO 155
125 PRINT "Tare Failed"
130 SLEEP 1000
150 GOTO 90
155 PRINT "Fill Container"
160 IF MotionA=0 THEN GOTO 160
170 SLEEP 3000
180 SLEEP 200
190 IF SPfeeding=0 and SPtolerance=0 THEN PRINT "Too Much Fill"
192 IF SPfeeding=1 and SPtolerance=0 THEN PRINT "More Fill"
193 IF SPtolerance=1 THEN PRINT "Fill In Tolerance"
194 IF SPtolerance=1 and MotionA=0 THEN GOTO 200
195 GOTO 180
200 Print "Filling Complete"
220 SLEEP 3000
230 PRINT "Remove Container"
240 DiscreteOut=1
260 IF NetWt > 0.0 THEN GOTO 260
270 DiscreteOut=0
280 PRINT "Completed"
290 SLEEP 3000
293 ClearTareA=1
294 IF ClearTareA=1 GOTO 294
300 GOTO 60
```

Rate-based Setpoint Auto-Preact

JagBASIC application can set up an Auto-Preact setpoint. In an auto-preact setpoint, the preact weight is automatically adjusted based on the rate that material is being filled or discharged from a hopper and an auto-preact time value. Whenever the JAGXTREME terminal calculates a new rate value, it adjusts the preact weight for the setpoint based on the rate and the auto-preact time value.

Auto-preact time value is the number of seconds it takes for the gate to close and the filling to complete once the JAGXTREME terminal detects that coincidence weight - preact weight has been reached. The auto-preact time is stored in shared data variable "spn06", where "n" is the number of the setpoint. The JAGXTREME terminal automatically learns the best auto-preact time by adjusting the value based on the error weight in each trial. Once a setpoint reaches coincidence value, the JAGXTREME calculates the difference between the setpoint coincidence value and the actual weight in the weigh hopper once the hopper reaches a "no motion" state. This difference is the error for the last trial. The JAGXTREME adjusts the auto-preact time by a value proportional to the error in the last trial and the sum of the errors over all trials.

To setup the auto-preact, the JagBASIC application must set the setpoint target to either "H" for a gross weight setpoint or "M" for net weight setpoint. You should initialize the auto-preact time value to your best guess of the preact time to minimize the number of trials it takes for the JAGXTREME terminal to learn and adjust to the best preact time. The other fields of the auto-preact setpoint are the same as in a standard setpoint.

With the JagBASIC ladder commands, you can use the setpoint feeding output to generate a discrete output for opening and closing a feed gate.

Example

```
REM // ************************************************
REM //  Setup Rate
REM // ************************************************
DEFSHR unit$,wt103
DEFSHR rateDisplay%,cs188
DEFSHR auxDisplay%,cs189
DEFSHR rateUnit%,cs101
DEFSHR period$,cs104
DEFSHR sample%,cs105
DEFSHR freq%,cs106
DEFSHR setup%,t_697
IF unit$="lb" THEN rateUnit%=1 ELSE rateUnit%=2
period$="S":rate = weight units per second
sample%=2:REM rate averaged over last two seconds
freq%=2:REM rate calculation frequency 1=1 sec;2=5 sec;2=half-second
rateDisplay%=0:auxDisplay%=0:REM turn-off rate display
setup%=1

REM ************************************************
REM Setpoint #1
REM Filling Setpoint using Auto preact
REM ************************************************
DEFSHR coincidence#,sp105
DEFSHR autopreact#,sp106
DEFSHR target,sp103
DEFSHR filling%,sp186
DEFSHR enable%,sp102
DEFSHR latching%,sp187
DEFSHR latched%,sp188
enable%=1
latching%=1
filling%=0
target$="H"
latched%=0
coincidence#=1000.0:REM weight
autopreact#=1.2:REM seconds
NEWLADDER
RUNGMOV s_210,p_500
DEFSHR setpoint%,t_698
setpoint%=1
```

Simple Truck In-Out

This JagBASIC program is used for a simple truck inbound/outbound application.

```
10 DEFSHR gross#,wt110:REM gross weight
20 DEFSHR unit$,wt103:REM weight units
30 DEFSHR stopEnable%,bas89
40 DEFSHR keyboards%,bas10
50 DEFSHR motion%,s_200
60 stopEnable%=0
70 keyboards%=3
80 password$="555555"
100 REM main menu
110 PRINT "IN = 1   OUT = 4"
```

```
120 GOSUB 3000
130 IF k$="1" THEN GOTO 1000
140 IF k$="4" THEN GOTO 2000
150 IF k$="7" THEN GOTO 5000
160 IF k$="8" THEN GOTO 6000
170 IF k$="9" THEN GOTO 7000
180 GOTO 120
1000 PRINT "Inbound?"
1005 GOSUB 3000
1006 IF k$<>CHR$(8) THEN GOTO 100
1010 IF gross#<10.0 THEN PRINT "SCALE EMPTY":GOTO 1180
1020 PRINT "Register #"
1030 OPEN "inbound.dat" for random as #1 len=10
1040 FIELD #1,8 as inwght$,2 as lfcr$
1050 reg%=999
1060 FOR x% = 1 to 50
1070 GET #1,x%
1080 IF inwght$="00000000" THEN reg%=x%: x%=50
1090 NEXTx%
1100 IF reg%=999 THEN PRINT "Memory Full":GOTO 1170
1105 IF motion%=1 THEN PRINT "Scale In Motion"
1106 IF motion%=1 THEN GOTO 1106
1110 LSET lfcr$=CHR$(13)+CHR$(10)
1120 RSET inwght$=mkd$(gross#)
1130 LPRINT "Register # ";reg%
1140 LPRINT date$;"  ";time$;"  ";gross#;" ";units$;" IN"
1150 PRINT "Register # ";reg%
1160 PUT #1,reg%
1170 CLOSE #1
1180 GOSUB 3000
1190 GOTO 100
2000 PRINT "Outbound?"
2005 GOSUB 3000
2006 IF k$<>CHR$(8) THEN GOTO 100
2010 INPUT "Enter Register",reg%
2030 IF reg%<1 or reg%>50 THEN GOTO 2010
2040 PRINT "Register # ";reg%
2045 SLEEP 1000
2050 OPEN "inbound.dat" for random as #1 len=10
2060 FIELD #1,8 as inwght$,2 as lfcr$
2070 GET #1,reg%
2080 IF inwght$="00000000" THEN PRINT "Register Empty":close #1:GOTO 2220
2090 IN#=cvd(inwght$)
2100 LSETlfcr$=CHR$(13)+CHR$(10)
2110 RSETt inwght$="00000000"
2120 PUT #1,reg%
2130 CLOSE #1
2135 IF motion%=1 THEN PRINT "Scale In Motion"
2136 IF motion%=1 THEN GOTO 2136
2140 IF in#>gross# THEN finalGross#=in#:tare#=gross#:GOTO 2160
2150 tare#=in#:finalGross#=gross#
2160 net#=finalGross#-tare#
2170 PRINT using "NET__######.#_!!";net#;unit$
2180 LPRINT DATE$+"  "+TIME$
2190 LPRINT using "NET____######.#_!!";net#;unit$
2200 LPRINT using "GROSS__######.#_!!";finalGross#;unit$
2210 LPRINT using "TARE___######.#_!!";tare#;unit$
```

```
2220 GOSUB 3000
2230 GOTO 100
3000 REM get key
3010 k$=inkey$
3020 IF k$="" THEN GOTO 3010
3030 RETURN
5000 PRINT "View Regs?"
5005 GOSUB 3000
5010 IF k$<>CHR$(8) THEN GOTO 100
5020 INPUT "Enter Password";pw$
5030 IF password$<>pw$ THEN GOTO 100
5040 OPEN "inbound.dat" for random as #1 len=10
5050 field #1,8 as inwght$,2 as lfcr$
5060 PRINT "Printout? Y=3"
5065 GOSUB 3000
5080 IF k$="3" THEN LPRINT "Reg  Stored Weight"
5090 FOR x%=1 to 50
5100 GET #1,x%
5110 IF inwght$="00000000" THEN GOTO 5150
5120 PRINT using "##__######.#";x%;cvd(inwght$)
5130 IF k$="3" THEN LPRINT using "##__######.#";x%;cvd(inwght$)
5140 SLEEP 1000
5150 NEXT x%
5160 CLOSE #1
5170 GOTO 100
6000 PRINT "Reset Regs?"
6005 GOSUB 3000
6010 IF k$<>CHR$(8) THEN GOTO 100
6020 INPUT "Enter Password"; pw$
6030 IF password$<>pw$ THEN GOTO 100
6040 open "inbound.dat" FOR OUTPUT AS#1
6050 for x%=1 to 50
6060 PRINT #1,"00000000"
6070 NEXT x%
6080 CLOSE #1
6090 PRINT "Reset Complete"
6100 SLEEP 2000
6110 GOTO 100
7000 PRINT "Exit?"
7005 GOSUB 3000
7010 IF k$<>CHR$(8) THEN GOTO 100
7020 INPUT "Enter Password";pw$
7030 IF password$<>pw$ THEN GOTO 100
7040 keyboards%=0
7050 END
```

Truck Inbound-Outbound

This application records the weight of a truck when it arrives at a plant, calculates the net weight of the truck when it leaves the plant, and updates tallies as directed by the operator. It uses up to two scales connected to a JAGXTREME terminal. Typical uses of this application are to record and tally the amount of:

- Asphalt loaded at an asphalt plant.
- Grain delivered to a grain elevator.
- Trash delivered to a trash dump.

This application uses the JagBASIC preprocessor, which uses the program source code listed here as input and generates the output file which runs on the JAGXTREME terminal.

Printing Tickets

This application prints a ticket after each truck inbound or outbound processing operation using the Demand Custom Print #3 connection. The operator must assign a serial port to this connection using the "CONFIG SERIAL" menu in the JAGXTREME terminal setup menus. The operator must also use the "CONFIG TEMPLATE" menu to setup the ticket format. This application sets the print literals as follows:

Literal 1	Header 1. Set up using Memory Key.
Literal 2	Header 2. Set up using Memory Key.
Literal 3	Net Weight.
Literal 4	Tare Weight.
Literal 5	Gross Weight.
Literal 6	Truck ID.
Literal 7	Tally 1 ID.
Literal 8	Tally 1 Weight Value.
Literal 9	Tally 2 ID.
Literal 10	Tally 2 Weight Value.
Literal 11	Tally 3 ID.
Literal 12	Tally 3 Weight Value.
Literal 13	Tally 4 ID.
Literal 14	Tally 4 Weight Value.
Literal 15	Tally 5 ID.
Literal 16	Tally 5 Weight Value.
Literal 17	Tally 6 ID.
Literal 18	Tally 6 Weight Value.

Processing Modes

This application has two processing modes: File Maintenance and Truck Inbound/Outbound. The operator presses the Esc key to switch between them.

File Maintenance Processing

The application maintains two files: the Truck File and the Tally File. The number of records stored in each file is limited to 1000 records since the JAGXTREME RAMDISK is 64K bytes. These files are random-access files with the records stored alphabetically by ID. The application quickly retrieves records from the files through a binary search. In File Maintenance processing, the operator can perform the following operations:

- Edit the Truck File
- Print the Contents of the Truck File
- List the Truck IDs
- Edit the Tally File
- Print the contents of the Tally File

Truck File
The Truck File has one record for each truck. The application weight units are the same as the scale's primary calibration units. The application does not support unit switching. Each record in the Truck File is 26 bytes long and has the following format:

Truck ID	8 characters
Tare Weight in ASCII	6 characters
Total Weight in ASCII	8 characters
Tare Type P/T	1 character
Truck In Plant Y/N	1 character
Line Feed/Carriage Return	2 characters
	26 characters

The Total Weight is the sum of the truck's net weights in all its trips to the plant. The truck's Tare Type is either "P" or "T".

"P" indicates that the operator entered the tare through the keyboard.

"T" indicates that the operator entered the tare by weighing the truck on the scale.

Tally File
The Tally File has one record for each tally that the operator records. Each record in the Tally File is 20 bytes long and has the following format:

Tally ID	8 characters
Tally Weight in ASCII	10 characters
Line Feed/Carriage Return	2 characters
	20 characters

Truck Inbound/Outbound Processing
In truck inbound processing, the application prompts the operator to enter the truck ID. If the truck ID does not exist in the Truck File, the application creates a new record in the file. The application records the inbound weight of the truck in the Truck File. The application sets the values in Literals 5 and 6; and blanks Literals 3, 4 and 7 through 18. The application issues the command to print a ticket through the Demand Custom Print #3 connection.

In truck outbound processing, the application prompts the operator to enter the truck ID. The application retrieves the inbound weight of the truck from the Truck File and calculates the net weight of the truck. The application prompts the user to enter up to six tally IDs and adds the net weight to each tally. The application sets the print literals and issues the command to print the ticket.

Operations Program

This program code executes the steps needed to carry out the inbound/outbound application.

```
REM ***********************************************
REM  TRUCK IN/OUT PROGRAM
REM ***********************************************
IF OP%=0 THEN GOTO Initialize
IF OP%=2 THEN GOTO TruckInOutStart ELSE GOTO MaintenanceStart
Initialize:
DIM Gross#(2):DEFSHR Gross#(1),wt110:DEFSHR Gross#(2),wt210
DIM Motion%(2):DEFSHR Motion%(1),s_200:DEFSHR Motion%(2),s_208
DIM Select%(2):DEFSHR Select%(1),t_6c0:DEFSHR Select%(2),t_6c1
DIM Zero%(2):DEFSHR Zero%(1),t_693:DEFSHR Zero%(2),t_6a3
DEFSHR Unit$,wt103
DEFSHR stopEnable%,bas89:stopEnable%=1
DEFSHR selKey,bas87:selKey=0:DEFSHR escKey,bas86:escKey=0
DEFSHR numScales,jag15:DEFSHR CustomPrint3%,t_6c5

DIM L$(18):DEFSHR L$(1),lit01:DEFSHR L$(2),lit02:DEFSHR L$(3),lit03
DEFSHR L$(4),lit04:DEFSHR L$(5),lit05:DEFSHR L$(6),lit06
DEFSHR L$(7),lit07:DEFSHR L$(8),lit08:DEFSHR L$(9),lit09
DEFSHR L$(10),lit10:DEFSHR L$(11),lit11:DEFSHR L$(12),lit12
DEFSHR L$(13),lit13:DEFSHR L$(14),lit14:DEFSHR L$(15),lit15
DEFSHR L$(16),lit16:DEFSHR L$(17),lit17:DEFSHR L$(18),lit18

REM ***********************************************
REM ***********************************************
REM  TRUCK INBOUND/OUTBOUND OPERATIONS
REM ***********************************************
REM ***********************************************
TruckInOutStart:
OP%=1:Scl%=1:Select%(1)=1
NextTruck:
REM troff
FOR i%=3 to 18
L$(i%)=" "
NEXT i%
REM tron
CustomPrint3%=0

INPUT "Truck";M$
REM ***** ZERO SCALE *****
IF M$=CHR$(7) THEN Zero%(Scl%)=1:GOTO NextTruck
REM ***** PROCESS TRUCK *****
IF M$<>CHR$(1) THEN GOSUB ProcessTruck:GOTO NextTruck
REM ***** SELECT SCALE *****
IF numScales=1 THEN GOTO NextTruck
IF Scl%=1 THEN Scl%=2 ELSE Scl%=1
Select%(Scl%)=1:GOTO NextTruck
```

```
REM ***********************************************
REM     PROCESS TRUCK
REM ***********************************************
ProcessTruck:
IF M$="" THEN RETURN
GOSUB CheckIDString:truckID$=M$:L$(6)=M$
GOSUB GetWgt:M$=STR$(Weight#):GOSUB SetToWidth8:L$(5)=M$

GOSUB OpenTruck:LSET TrkID$=truckID$:ON ERROR GOTO NewInboundTruck:GET #2
IF TIP$="Y" THEN GOTO OutboundTruck

REM ***********************************************
REM     PROCESS INBOUND TRUCK
REM ***********************************************
InboundTruck:
PRINT "InBound ";truckID$:SLEEP 1000
IF TTyp$="P" THEN L$(5)=TW$ ELSE RSET TW$=RIGHT$(L$(5),8)
GOTO DoneInbound

REM ***********************************************
REM     NEW INBOUND TRUCK ID
REM ***********************************************
NewInboundTruck:
IF ERR()<>6 THEN LPRINT ERR();" ";ERL():END
PRINT "New ID ";truckID$;"?":GOSUB GetKey
IF C$<>CHR$(8) THEN CLOSE #2:RETURN
RSET TW$=RIGHT$(L$(5),8)
LSET TTyp$="T":LSET TrkID$=truckID$:RSET TTot$="         0"

DoneInbound:
RSET TIP$="Y":LSET cr$=CHR$(13)+CHR$(10):PUT #2
GOSUB PrintHeader:LPRINT "Inbound Truck  ";truckID$
LPRINT USING "GROSS_WT____##########._!!";VAL(L$(5));Unit$:LPRINT ""
CustomPrint3%=1:CLOSE #2:RETURN

REM ***********************************************
REM     PROCESS OUTBOUND TRUCK
REM ***********************************************
OutboundTruck:
NetWt#=Weight#-VAL(TW$):M$=STR$(NetWt#):Width%=8:GOSUB
SetToWidth:L$(3)=M$
M$=TW$:GOSUB SetToWidth8:L$(4)=M$
Weight#=NetWt#+VAL(TTot$)
M$=STR$(Weight#):Width%=10:GOSUB SetToWidth:RSET TTot$=M$
RSET TIP$="N":LSET cr$=CHR$(13)+CHR$(10):PUT #2:CLOSE #2

REM ***********************************************
REM  HEY, MR. TALLY MAN, TALLY ME BANANAS.
REM ***********************************************
t%=0:GOSUB OpenTally
```

```
MoreTallies:
INPUT "Enter Tally ID",M$:IF M$=" " OR M$="" THEN GOTO DoneTallies
GOSUB CheckIDString:tallID$=M$
ON ERROR GOTO NewTally
LSET TallyID$=tallID$:GET #1

REM *** FOUND EXISTING TALLY
M$=STR$(VAL(Tally$)+NetWt#):Width%=10:GOSUB SetToWidth:RSET Tally$=M$
GOTO PutTally

NewTally:
IF ERR()<>6 THEN LPRINT ERR();" ";ERL():END
PRINT "AddNew ";tallID$;"?":GOSUB GetKey:IF C$<>CHR$(8) THEN GOTO MoreTallies
M$=STR$(NetWt#):Width%=10:GOSUB SetToWidth:RSET Tally$=M$

PutTally:
L$(t%*2+7)=tallID$:L$(t%*2+8)=Tally$:REM *** SET LITERALS FOR CUSTOM PRINT
LSET TallyID$=tallID$:LSET cr$=CHR$(13)+CHR$(10):PUT #1

NextTally:
t%=t%+1:IF t%<6 THEN GOTO MoreTallies

DoneTallies:
GOSUB PrintHeader:LPRINT "Outbound Truck  ";truckID$;CHR$(10)
LPRINT USING "GROSS_WT____##########._!!";VAL(L$(5));Unit$
LPRINT USING "TARE_WT_____##########._!!";VAL(L$(4));Unit$
LPRINT USING "NET_WT______##########._!!";VAL(L$(3));Unit$
LPRINT "":i%=0
MoreTallyPrint:
IF i%>=t%*2 THEN LPRINT CHR$(10):CLOSE #1:CustomPrint3%=1:RETURN
LPRINT USING "!!!!!!!!__############._!!";L$(i%+7);VAL(L$(i%+8));Unit$
i%=i%+2:GOTO MoreTallyPrint


REM ************************************************
REM ************************************************
REM          TRUCK FILES MAINTENANCE MAIN MENU
REM ************************************************
REM ************************************************
MaintenanceStart:
IF OP%=4 THEN GOTO MenuPrintTruck
IF OP%=5 THEN GOTO MenuListTruck
IF OP%=6 THEN GOTO EditTallyMenu
IF OP%=7 THEN GOTO MenuPrintTally
IF OP%=8 THEN GOTO MenuSendFiles

Maintenance:
OP%=2:PRINT "Edit Truck File":GOSUB GetKey
IF C$=CHR$(8) THEN GOSUB EditTruckFile:GOTO Maintenance

MenuPrintTruck:
OP%=2:PRINT "Print Truck File":GOSUB GetKey
IF C$=CHR$(8) THEN GOSUB PrintTrucks

MenuListTruck:
OP%=2:PRINT "List Truck IDs":GOSUB GetKey
IF C$=CHR$(8) THEN GOSUB ListTruckID
```

```
EditTallyMenu:
OP%=2:PRINT "Edit Tally File":GOSUB GetKey
IF C$=CHR$(8) THEN GOSUB EditTallyFile:GOTO EditTallyMenu

MenuPrintTally:
OP%=2:PRINT "Print Tally File":GOSUB GetKey
IF C$=CHR$(8) THEN GOSUB PrintTallyList

MenuSendFiles:
OP%=2:PRINT "Send Data Files":GOSUB GetKey
IF C$=CHR$(8) THEN GOSUB SendFiles
GOTO Maintenance

REM ***********************************************
REM         EDIT THE TRUCK FILE
REM ***********************************************
EditTruckFile:
OP%=3:INPUT "Enter Truck ID",M$
GOSUB OpenTruck:ON ERROR GOTO NewTruck:r%=0
IF M$<>" " AND M$<>"" THEN GOTO SearchTruckID

LookNextID:
IF EOF(2) THEN CLOSE #2:PRINT "End Of File":SLEEP 2000:RETURN
r%=r%+1:GET #2,r%:PRINT "Truck ";TrkID$;"?":GOSUB GetKey
IF C$=CHR$(8) THEN truckID$=TrkID$:GOTO EditRecord ELSE GOTO LookNextID

SearchTruckID:
GOSUB CheckIDString:truckID$=M$:PRINT "Search ";truckID$:SLEEP 1000
LSET TrkID$=truckID$:GET #2

EditRecord:
PRINT "Edit ";truckID$;"?":GOSUB GetKey:IF C$=" " THEN GOTO DeleteTruck
IF C$<>CHR$(8) THEN GOTO EndTruckEdit
PRINT "Outbound?    Y/N",C$:GOSUB GetKey
IF C$="Y" THEN RSET TIP$="Y" ELSE RSET TIP$="N"
GOTO SetTare

DeleteTruck:
PRINT "Delete ";truckID$;"?":GOSUB GetKey:IF C$=" " THEN GOTO EditRecord
IF C$=CHR$(8) THEN PRINT "Deleting ";truckID$:SLEEP 1000:DELREC #2
GOTO EndTruckEdit

NewTruck:
IF ERR()<>6 THEN LPRINT ERR();" ";ERL():END
PRINT "Add ";truckID$;"?":GOSUB GetKey:IF C$<>CHR$(8) THEN GOTO EndTruckEdit
PRINT "Adding ";truckID$ :LSET TrkID$=truckID$
RSET TW$="        ":RSET TTot$="          ":LSET TTyp$=" ":RSET TIP$="N"

SetTare:
PRINT "Tare Type?   P/T":GOSUB GetKey
IF C$=CHR$(4) OR C$="T" THEN RSET TTyp$="T":GOSUB GetWgt:GOTO SetTot
IF C$="P" THEN RSET TTyp$="P":INPUT "Tare Wt:",Weight#:GOTO SetTot
PRINT "Invalid Type":SLEEP 2000:GOTO SetTare
```

```
SetTot:REM ***** SET TOTAL WEIGHT ******
M$=STR$(Weight#):GOSUB SetToWidth8:RSET TW$=M$
INPUT "Total:",Weight#
M$=STR$(Weight#):Width%=10:GOSUB SetToWidth:RSET TTot$=M$
LSET TrkID$=truckID$:LSET cr$=CHR$(13)+CHR$(10):PUT #2

EndTruckEdit:
CLOSE #2:RETURN

REM ***********************************************
REM         EDIT THE Tally ID FILE
REM ***********************************************
EditTallyFile:
OP%=6:INPUT "Enter Tally ID",M$
GOSUB OpenTally:ON ERROR GOTO MakeNewTally:r%=0
IF M$<>" " AND M$<>"" THEN GOTO SearchTallyID

LookTally:
IF EOF(1) THEN CLOSE #1:PRINT "End Of File":SLEEP 2000:RETURN
r%=r%+1:GET #1,r%:PRINT "Tally ";TallyID$;"?":GOSUB GetKey
IF C$=CHR$(8) THEN talID$=TallyID$:GOTO EditTallyRecord ELSE GOTO LookTally

SearchTallyID:
GOSUB CheckIDString:talID$=M$:PRINT "Search ";talID$:SLEEP 1000
LSET TallyID$=talID$:GET #1

EditTallyRecord:
PRINT "Edit ";talID$;"?":GOSUB GetKey
IF C$=CHR$(8) THEN GOTO WriteTallyTotal
IF C$<>" " THEN GOTO EndTallyEdit

DeleteTally:
PRINT "Delete ";talID$;"?":GOSUB GetKey
IF C$=" " THEN GOTO EditTallyRecord
IF C$=CHR$(8) THEN PRINT "Deleting ";talID$:SLEEP 1000:DELREC #1
GOTO EndTallyEdit

MakeNewTally:
IF ERR()<>6 THEN LPRINT ERR();" ";ERL():END
PRINT "Add ";talID$;"?":GOSUB GetKey:IF C$<>CHR$(8) THEN GOTO EndTallyEdit
PRINT "Adding ";talID$:SLEEP 1000:GOTO WriteTallyTotal

WriteTallyTotal:
LSET TallyID$=talID$:Input "Total:",Weight#
M$=STR$(Weight#):Width%=10:GOSUB SetToWidth:RSET Tally$=M$
LSET cr$=CHR$(13)+CHR$(10):PUT #1

EndTallyEdit:
CLOSE #1:RETURN
```

```
REM ***********************************************
REM         PRINT THE TRUCK FILE
REM ***********************************************
PrintTrucks:
OP%=4:PRINT "Clear Total? N/Y":GOSUB GetKey:PRINT "Printing..."
GOSUB PrintHeader:LPRINT "Truck Report"
LPRINT "Truck ID";TAB(18);"Tare Weight";TAB(38);"Total";TAB(47);"Outbound"
LPRINT STRING$(54,"="):GOSUB OpenTruck:r%=0
WHILE NOT EOF(2)
r%=r%+1:GET #2,r%:LPRINT TrkID$;TAB(15);
IF C$="Y" OR C$=CHR$(6) THEN RSET TTot$="         0"
LPRINT USING "########._!!_!";VAL(TW$);Unit$;TTyp$;
LPRINT USING "__##########._!!____!";VAL(TTot$);Unit$;TIP$
IF C$="Y" OR C$=CHR$(6) THEN PUT #2,r%
WEND
LPRINT r%;" Trucks":PRINT r%;" Trucks":SLEEP 2000:CLOSE #2:RETURN

REM ***********************************************
REM         PRINT LIST OF TRUCKS
REM ***********************************************
ListTruckID:
OP%=5:GOSUB PrintHeader
LPRINT "Truck ID List":LPRINT STRING$(25,"="):GOSUB OpenTruck:r%=0
PrintNextTruck:
IF NOT EOF(2) THEN r%=r%+1:GET #2,r%:LPRINT TrkID$:GOTO PrintNextTruck
LPRINT r%;" Trucks":PRINT r%;" Trucks":SLEEP 2000:CLOSE #2:RETURN

REM ***********************************************
REM         PRINT LIST OF TALLIES
REM ***********************************************
PrintTallyList:
OP%=7:GOSUB PrintHeader
LPRINT "Tally";TAB(18);"Total":LPRINT STRING$(25,"="):GOSUB OpenTally:r%=0
PrintNextTally:
IF EOF(1) THEN GOTO PrintTallyDone
r%=r%+1:GET #1,r%:LPRINT TallyID$;
LPRINT USING "___##########._!!";VAL(Tally$);Unit$:GOTO PrintNextTally
PrintTallyDone:
LPRINT r%;" Tallies":PRINT r%;" Tallies":SLEEP 2000:CLOSE #1:RETURN

REM ***********************************************
REM    SEND FILES TO HOST USING ZMODEM
REM ***********************************************
SendFiles:
OP%=8:PRINT "Files To Host":GOSUB GetKey:IF C$<>CHR$(8) THEN GOTO
ReceiveFiles
PRINT "Are You Sure?":GOSUB GetKey:IF C$<>CHR$(8) THEN RETURN
PRINT "":SZ "TRUCK":SZ "TALLY":RETURN
ReceiveFiles:
PRINT "Files From Host":GOSUB GetKey:IF C$<>CHR$(8) THEN RETURN
PRINT "Are You Sure?":GOSUB GetKey:IF C$=CHR$(8) THEN RZ ELSE RETURN
PRINT "SORTING FILES":GOSUB OpenTruck:SORTREC #2,TrkID$:CLOSE #2
GOSUB OpenTally:SORTREC #1,TallyID$:CLOSE #1:RETURN
```

```
REM ************************************************
REM                    GET WEIGHT OF TRUCK
REM ************************************************
GetWgt:
Scl%=1:C$="A":If numScales=1 THEN GOTO CheckMotion
PRINT "Scale?   A/B":GOSUB GetKey
IF C$="B" OR C$=CHR$(5) THEN Scl%=2:C$="B" ELSE C$="A"
Select%(Scl%)=1
CheckMotion:
PRINT "Weighing Scale ";C$:SLEEP 1000
IF Motion%(Scl%)=1 THEN PRINT "Scale In Motion":SLEEP 250:GOTO CheckMotion
Weight#=Gross#(Scl%):RETURN

REM ************************************************
REM        OPEN TRUCK FILE
REM ************************************************
OpenTruck:
OPEN "TRUCK" FOR RANDOM AS #2 LEN=30
FIELD #2,8 AS TrkID$,8 AS TW$,1 AS TTyp$,10 AS TTot$,1 AS TIP$,2 AS cr$
INDEXED #2,TrkID$:RETURN

REM ************************************************
REM        OPEN TALLY FILE
REM ************************************************
OpenTally:
OPEN "TALLY" FOR RANDOM AS #1 LEN=20
FIELD #1,8 AS TallyID$,10 AS Tally$,2 AS cr$
INDEXED #1,TallyID$:RETURN

REM ************************************************
REM  Print Report Header
REM ************************************************
PrintHeader:
LPRINT CHR$(10)+CHR$(10):LPRINT L$(1):LPRINT L$(2)+CHR$(10)
LPRINT DATE$;TAB(19);TIME$+CHR$(10):RETURN

REM ************************************************
REM        GET A KEY
REM ************************************************
GetKey:
REM troff
C$=INKEY$:IF C$<>"" THEN GOTO GetKey
GetKey1:
C$=INKEY$:IF C$="" THEN GOTO GetKey1
REM tron
IF C$>="a" AND C$<="z" THEN C$=CHR$(ASC(C$)-32)
IF C$=CHR$(2) THEN RESTART ELSE RETURN

REM ************************************************
REM                    CHECK TERMINATING CHARACTERS ON STRING
REM     BLANK FILL ID TO WIDTH 8
REM                    CAPITALIZE ID
REM ************************************************
CheckIDString:
C$=RIGHT$(M$,1):IF C$=CHR$(2) THEN RESTART
IF C$<CHR$(8) THEN M$=LEFT$(M$,LEN(M$)-1)
AddBlank:
```

```
IF LEN(M$)<8 THEN M$=M$+" ":GOTO AddBlank
A$=M$:M$=""
FOR i%=1 TO 8
C$=MID$(A$,i%,1)
IF C$>="a" AND C$<="z" THEN M$=M$+CHR$(ASC(C$)-32) ELSE M$=M$+C$
NEXT i%
RETURN

REM ***********************************************
REM  RIGHT SHIFT NUMERIC STRING TO SPECIFIED WIDTH
REM ***********************************************
SetToWidth8:
Width%=8
SetToWidth:
IF LEN(M$)<Width% THEN M$=" "+M$:GOTO SetToWidth
IF LEN(M$)>Width% THEN M$=LEFT$(M$,Width%)
RETURN
```

Multiple Ingredient Formulation (Manual Batching)

JagBASIC can be applied to a Multiple Ingredient Formulation application (Manual batching). This example uses various JagBASIC programming techniques for operations such as maintaining data files, acquiring weight data, and controlling output to the terminal lower display. PCJagBASIC must process this program before it can be downloaded and run on a JAGXTREME terminal. PCJagBASIC is a development tool, which simplifies the development and maintenance of JagBASIC programs. This program was designed to work with a two-scale system.

File Maintenance

The Multiple Ingredient Formulation Application uses 2 files for data storage, Material.dat and Recipe.dat. File maintenance enables the operator to add, delete, and edit records of recipes and materials. These files are random-access files with records stored alphabetically by ID.

The material file has 2 fields, MaterialID and Inventory. The application updates the inventory as a material is used. The operator may update inventory by using the program to edit the material file.

The recipe file has 5 fields, RecipeID, MaterialID, ScaleID, Amount, and Tolerance. The Recipe ID is composed of a user assigned name, and a generated index number. The index number is assigned by the JagBASIC program and is used to determine the order in which to add the ingredients. The first ingredient has a Recipe ID of "RecipeName00"; the second ingredient has an ID of "RecipeName01", and so on.

In the Recipe file, the MaterialID identifies which material to add to the batch, and the ScaleID indicates which scale to add the material to. The Amount indicates how much material to add, and Tolerance is the acceptable amount of error in the delivery.

Material File	
MaterialID	10 characters
Inventory	8 characters

Recipe File	
RecipeID	12 characters
MaterialID	10 characters
ScaleID	1 characters
Amount	8 characters
Tolerance	8 characters

In order to create a recipe and run a batch, the operator must first create a list of materials. The operator must use the select and enter keys on the terminal to choose "File_Maint" and then "Material_File" from the menus. The operator may then follow the prompts to edit, add, or delete a material from the file. After several materials are created, the operator may create a recipe. The select and enter keys should be used to choose "File_Maint" and then "Recipe_File" from the menus. The operator may then follow the prompts to edit, add, or delete a recipe from the file.

The application also allows the operator to print reports detailing the amount of materials available or the components of the recipes. Samples of each report follow.

```
==============================
|   Materials Report        |
| 07-21-2000    08:24:17    |
==============================
|  ID        Inventory      |
==============================
   Chocolate   98.04
   Eggs        20
   Nutmeg      32.5
   Flour       167.78
   Water       50.23
==============================

================================================
|   Recipe Report   07-21-2000 08:31:56        |
================================================
| Recipe     Material ID   Scale  Amount     Tol.|
================================================
 Cake:
          Flour          A      5          1
          Milk           B      2          1
          Chocolate      B      3.5        1
          Eggs           A      1          .5
 Paste:
          Flour          B      6          .25
          Water          B      3.5        .5
================================================
```

OPERATIONS

To begin a batch, the operator should choose "Run_Recipe" from the main menu. Once a recipe is selected, a confirmation prompt will appear. The operator should choose "Y" to run the batch. If the required materials are not present in sufficient quantity, as determined by examining the inventory in the material file, an error message will appear on the lower display of the terminal and the patch will terminate. A material's inventory can be modified by selecting "Edit_Material" under the "File_Maint" and "Material_File" menus.

Once the batch begins, the ID of the first ingredient, along with the quantity remaining to be added, will appear on the lower display of the terminal. This message will begin to blink once that material is within tolerance. The terminal's discrete outputs are also an indication that the material is within tolerance. Discrete output 1 is on while the material is out of tolerance and discrete output 2 is on while the material is within tolerance. To move on to the next ingredient, the operator must push the enter key on the terminal. The materials inventory file is updated after each ingredient is added. After the batch is complete, an audit trail report is printed.

```
================================
|   Recipe Audit Trail       |
| 07-21-2000    09:12:56     |
================================
  Recipe: Cake
  Material      Target  Actual
================================
  Flour           5    5.13
  Milk            4    4.14
  Water         3.5    3.52
================================
              12.5    12.79
```

SOURCE FILE FOR PCJagBASIC

```
DefIn
DefOut
TableErrorOn

DEFSHR KeySrc#,bas10 : DEFSHR SelectEnable#,bas87
DEFSHR ManualStop#,bas89 : DEFSHR EscapeEnable#,bas86

DIM GWt#(2) : DEFSHR GWt#(1),wt110 : DEFSHR GWt#(2),wt211

DIM NWt#(2) : DEFSHR NWt#(1),wt111 : DEFSHR NWt#(2),wt211

DIM OnScale%(2) : DEFSHR OnScale%(1),t_6c0 : DEFSHR OnScale%(2),t_6c1
DIM MotionScale%(2) : DEFSHR MotionScale%(1),s_200 :DEFSHR
MotionScale%(2),s_208

DEFSHR TareScale%,t_6b0
DEFSHR ClearTareScale%,t_6b1

MATERIALFILE$ = "Material.dat"
RECIPEFILE$ = "Recipe.dat"
AUDITFILE$ = "Audit.txt"

' The Material file will have 2 fields.
' One for the Material ID, and another for Inventory.
DefTable Materials, "Material.dat"
   DefKey MaterialID$,10
   DefFld Inventory#
DefEnd

' The Recipe file will have 5 fields.  A recipe can be made up
' of 1-99 ingredients.  The user will enter in a recipe name,
' followed by the details (material name, scale, amount, tol.)
' of the first ingredient.  This information is saved under the
' RecipeID of "Name00".  The next ingredient is saved under the
' Recipe ID of "Name01", and the process continues.
```

```
DefTable Recipes, "Recipe.dat"
   DefKey RecipeID$,12
   DefFld MaterialID$,10
   DefFld ScaleID$,1
   DefFld Amount#
   DefFld Tolerance#
DefEnd

DISPLAY_ON# = 0.5 : DISPLAY_OFF# = 0.05
KeySrc# = 3 :  SelectEnable# = 0 : ManualStop# = 0 : EscapeEnable# = 0

RUN% = 1

Main_Menu:
   MENUSEL$ = "File_Maint"
   WHILE RUN% = 1
      INPUT "^^File_Maint,Run_Recipe,Reports,Exit_Program";MENUSEL$
      IF MENUSEL$ = "File_Maint" Then Gosub File_Maint
      IF MENUSEL$ = "Run_Recipe" Then Gosub Run_Recipe
      IF MENUSEL$ = "Reports" Then Gosub Reports
      IF MENUSEL$ = "Exit_Program" THEN RUN% = 0
   WEND
   KeySrc# = 2 :  SelectEnable# = 1 : ManualStop# = 1 : EscaleEnable# = 1
END

File_Maint:
   MENUSEL$ = "Material_File"
   WHILE RUN% = 1
      INPUT "^^Material_File,Recipe_File,Exit_File_Maint";MENUSEL$
      IF MENUSEL$ = "Material_File" Then Gosub Mat_File_Menu
      IF MENUSEL$ = "Recipe_File" Then Gosub Recipe_File_Menu
      IF MENUSEL$ = "Exit_File_Maint" THEN RUN% = 0
   WEND
   RUN% = 1
   MENUSEL$ = "File_Maint"
RETURN

Mat_File_Menu:
   MENUSEL$ = "Edit_Material"
   WHILE RUN% = 1
      INPUT "^^Edit_Material,Add_Material,Delete_Material,Exit_Mat_File";MENUSEL$
      IF MENUSEL$ = "Edit_Material" Then Gosub Edit_Material
      IF MENUSEL$ = "Add_Material" Then Gosub Add_Material
      IF MENUSEL$ = "Delete_Material" Then Gosub Delete_Material
      IF MENUSEL$ = "Exit_Mat_File" THEN RUN% = 0
   WEND
   MENUSEL$ = "Material_File"
   RUN% = 1
RETURN

Edit_Material:
   GOSUB Fill_Material_Array
   IF MaterialString$ = "-EMPTY-" THEN
      PRINT "--No Materials--" : SLEEP 1000
      RETURN
   ENDIF
   IF MaterialCount% > 1 THEN
      Menu$ = "^Edit ^"+MaterialString$
      MaterialID$ = ""
      INPUT Menu$;MaterialID$
```

```
 ELSE
      MaterialID$ = MaterialString$
   ENDIF
   PRINT "-Edit ";MaterialID$ : SLEEP 1000
   REM ' retrieve material and save in temp record
   REM ' then delete material.  If edit fails, restore material.
   FetchFrom Materials
   Temp$ = RTRIM$(MaterialID$)
   Temp# = Inventory#
   DeleteFrom Materials

   INPUT "^ID: ^ !!!!!!!!!!",MaterialID$
   If MaterialID$ = "" Then
      PRINT "Mat. Not Changed" : SLEEP 2000
      MaterialID$ = Temp$ : Inventory# = Temp#
      StoreTo Materials
      RETURN
   EndIf

   FetchFrom Materials

   IfTableError 6 Then
      REM ' record does not exist
      INPUT "^Invntry:^######.##",Inventory#
      StoreTo Materials
      PRINT "Material Edited" : SLEEP 2000
   ELSE
      PRINT "--Mat. Exists--" : SLEEP 1000
      PRINT "-Edit Canceled-" : SLEEP 1000
      MaterialID$ = Temp$ : Inventory# = Temp#
      StoreTo Materials
   Endif
RETURN

Add_Material:
   MaterialID$ = ""
   INPUT "^ID: ^ !!!!!!!!!!",MaterialID$
   If MaterialID$ = "" Then
      PRINT "-Mat. Not Added-" : SLEEP 2000
      RETURN
   EndIf

   If INSTR(MaterialID$," ") <> 0 THEN
      REM ' do not allow spaces in Material name
      Print "-Illegal Spaces-" : SLEEP 1000
      GOTO Add_Material
   ENDIF

   PRINT "Searching..."
   FetchFrom Materials

   IfTableError 6 Then
      REM ' Record does not exist
      Inventory# = 0
      INPUT "^Invntry:^######.##",Inventory#
      StoreTo Materials
      PRINT "-Material Added-" : SLEEP 2000
   ELSE
      REM 'can't add a record that already exists
      PRINT "--Mat. Exists--" : SLEEP 2000
```

```
 ENDIF
RETURN

Delete_Material:
   GOSUB Fill_Material_Array
   IF MaterialString$ = "-EMPTY-" THEN
      PRINT "--No Materials--" : SLEEP 1000
      RETURN
   ENDIF
   IF MaterialCount% > 1 THEN
      Menu$ = "^Del. ^"+MaterialString$
      INPUT Menu$;MaterialID$
   ELSE
      MaterialID$ = MaterialString$
   ENDIF

   Menu$ = "^Del."+MaterialID$+"^Y,N"
   ANS$ = "N"
   INPUT Menu$;ANS$

   IF ANS$ = "N" THEN RETURN

   DeleteFrom Materials
   PRINT "--Mat. Deleted--" : Sleep 2000
RETURN

Recipe_File_Menu:
   MENUSEL$ = "Edit_Recipe"
   WHILE RUN% = 1
      INPUT "^^Edit_Recipe,Add_Recipe,Delete_Recipe,Exit_Recipe_File";MENUSEL$
      IF MENUSEL$ = "Edit_Recipe" Then Gosub Edit_Recipe
      IF MENUSEL$ = "Add_Recipe" Then Gosub Add_Recipe
      IF MENUSEL$ = "Delete_Recipe" Then Gosub Delete_Recipe
      IF MENUSEL$ = "Exit_Recipe_File" THEN RUN% = 0
   WEND
   MENUSEL$ = "Recipe_File" : RUN% = 1
RETURN

Edit_Recipe:
   GOSUB Fill_Recipe_Array
   GOSUB Fill_Material_Array
   If RecipeString$ = "-EMPTY-" THEN
      Print "-- No Recipes --" : SLEEP 1000
      RETURN
   ENDIF
   IF RecipeCount% > 1 THEN
      Menu$ = "^Edit ^"+RecipeString$
      RecipeHead$ = ""
      INPUT Menu$;RecipeHead$
   ELSE
      RecipeHead$ = RecipeString$
      Print "Edit ";RecipeHead$ :SLEEP 1000
   ENDIF

   RecipeID$ = RecipeHead$ + "00"
   Tracker% = 0 : Done% = 0
   While Done%=0
      FetchFrom Recipes
      MaterialID$ = RTRIM$(MaterialID$)
      IfTableError 6 Then
         REM ' Record does not exist
```

```
            REM ' set defaults for Get_Material_Info
            MaterialID$ = "" : ScaleID$ = ""
            Amount# = 0 : Tolerance# = 0
         ENDIF
         GOSUB Get_Material_Info
         StoreTo Recipes

         ANS$ = "N"
         INPUT "^More ^Y,N";ANS$
         IF ANS$ = "N" THEN Done%=1
         GOSUB Get_Next_Ing
      Wend

      Done%=0
      While Done%=0
         FetchFrom Recipes
         IfTableError 6 Then
            REM ' record does not exist
            Done%=1
         Else
            DeleteFrom Recipes
            GOSUB Get_Next_Ing
         Endif
      Wend
   RETURN

   Add_Recipe:
      RecipeHead$ = ""
      INPUT "^ID: ^ !!!!!!!!!!",RecipeHead$
      If RecipeHead$ = "" Then
         PRINT "Recip. Not Added" : SLEEP 2000
         RETURN
      EndIf

      If INSTR(RecipeHead$," ") <> 0 THEN
         REM ' don't allow spaces in recipe name
         Print "-Illegal Spaces-" : SLEEP 1000
         GOTO Add_Recipe
      ENDIF

      Tracker% = 0
      RecipeID$ = RecipeHead$ + "00"
      PRINT "Searching..."
      FetchFrom Recipes
      IfTableError 6 Then
         GOSUB Fill_Material_Array
         Done%=0
         While Done%=0
            ScaleID$ = "A"
            Amount# = 0 : Tolerance# = 0
            GOSUB Get_Material_Info
            StoreTo Recipes
            ANS$ = "N"
            INPUT "^More ^Y,N";ANS$
            MaterialID$ = ""
            IF ANS$ = "N" THEN
               Done%=1
            ELSE
               GOSUB Get_Next_Ing
            ENDIF
         Wend
```

```
    ELSE
        PRINT "-Recipe Exists-" : SLEEP 2000
    ENDIF

RETURN

Delete_Recipe:
    GOSUB Fill_Recipe_Array
    If RecipeString$ = "-EMPTY-" THEN
        Print "-- No Recipes --" : SLEEP 1000
        RETURN
    ENDIF
    IF RecipeCount% > 1 THEN
        Menu$ = "^Del. ^"+RecipeString$
        RecipeHead$ = ""
        INPUT Menu$;RecipeHead$
    ELSE
        RecipeHead$ = RecipeString$
    ENDIF

    Menu$ = "^Del."+RecipeHead$+"^Y,N"
    ANS$ = "N"
    INPUT Menu$;ANS$

    IF ANS$ = "N" THEN RETURN

    Done% = 0 : Tracker% = 0
    RecipeID$ = RecipeHead$ + "00"
    DeleteFrom Recipes
    While Done%=0
        GOSUB Get_Next_Ing
        FetchFrom Recipes
        IfTableError 6 Then
            REM ' file not found, set flag to exit loop
            Done% = 1
        ELSE
            REM ' delete record
            DeleteFrom Recipes
        EndIf
    Wend
    PRINT "-Recipe Deleted-" : Sleep 2000
RETURN

Run_Recipe:
    GOSUB Fill_Recipe_Array
    If RecipeString$ = "-EMPTY-" THEN
        Print "-- No Recipes --" : SLEEP 1000
        RETURN
    ENDIF
    IF RecipeCount% > 1 THEN
        Menu$ = "^Run   ^"+RecipeString$
        RecipeHead$ = ""
        INPUT Menu$;RecipeHead$
    ELSE
        RecipeHead$ = RecipeString$
    ENDIF

    Menu$ = "^Run "+RecipeHead$+"^Y,N"
    ANS$ = "N"
    INPUT Menu$;ANS$
```

```
IF ANS$ = "N" THEN
   PRINT "  --Aborted--  " : Sleep 1000
   RETURN
ENDIF

GOSUB Check_Inventory  REM 'Set AbortRecipe%=1 if Inventory low
IF AbortRecipe% = 1 Then
   RETURN
ENDIF

REM 'Create file for new audit trail
OPEN AUDITFILE$ FOR OUTPUT AS #3
CLOSE #3

RecipeID$ = RecipeHead$ + "00"
ACC_TOTAL# = 0.0 : EXP_TOTAL# = 0.0
AuditRecord$ = "" : Tracker% = 0
FetchFrom Recipes

RUN% = 1
While RUN% = 1
   Temp$= STR$(Amount#)
   AuditRecord$ = MaterialID$ + " " + PADL$(Temp$,8," ")

   REM ' select scale to set focus on A or B
   Index% = 2
   If ScaleID$ = "A" Then Index% = 1
   OnScale%(Index%) = 1
   While OnScale%(Index%) = 1
   Wend

   TareScale% = 1
   While TareScale% = 1
   Wend

   Done% = 0
   OldTime# = TIMER()
   TimeIndex# = DISPLAY_ON#

   While Done% = 0
      REM ' code to blink the display
      If TimeIndex# = DISPLAY_OFF# THEN
         Message$ = ""
      ELSE
         Message$ = MaterialID$ + " " + STR$(Amount# - NWt#(Index%))
      ENDIF
      Print Message$

      Target# = ABS(NWt#(Index%) - Amount#)
      IF Target# < Tolerance# OR Target# = Tolerance# THEN
         REM ' code to set timer for display blink
         NewTime# = TIMER()
         If NewTime# - OldTime# > TimeIndex# THEN
              If TimeIndex# = DISPLAY_ON# THEN TimeIndex# = DISPLAY_OFF# ELSE
    TimeIndex# = DISPLAY_ON#
            OldTime# = TIMER()
         ENDIF
         SwitchON 2
         SwitchOFF 1
      ELSE
         TimeIndex# = DISPLAY_ON#
```

```
            SwitchON 1
            SwitchOFF 2
         ENDIF

         Key$ = INKEY$
         If Key$ = CHR$(8) THEN
            Done% = 1
            While MotionScale%(Index%) = 1
            Wend
         ENDIF

      Wend

      Delivered# = NWt#(Index%)
      Delivered$ = STR$(Delivered#)
      Delivered$ = PADL$(Delivered$,8," ")

      AuditRecord$ = AuditRecord$ + Delivered$
      OPEN AUDITFILE$ FOR APPEND AS #3
      WRITE #3,AuditRecord$
      CLOSE #3

      ACC_TOTAL# = ACC_TOTAL# + Delivered#
      EXP_TOTAL# = EXP_TOTAL# + Amount#

      REM ' adjust the inventory of the material
      GOSUB Adjust_Inventory
      GOSUB Get_Next_Ing
      FetchFrom Recipes
      IfTableError 6 Then
         RUN%=0
      Endif

   Wend
   SwitchOFF 1
   SwitchOFF 2

   REM ' print audit trail
   PRINT "Printing..."
   LPRINT "================================="
   LPRINT "|";TAB(6);"Recipe Audit Trail";TAB(32);"|"
   LPRINT "|";TAB(4);DATE$;TAB(18);TIME$;TAB(32);"|"
   LPRINT "================================="
   LPRINT TAB(4);"Recipe:";TAB(12);RecipeHead$
   LPRINT TAB(4);"Material";TAB(18);"Target";TAB(26);"Actual"
   LPRINT "================================="
   OPEN AUDITFILE$ FOR INPUT AS #3
   While EOF(3) = 0
      INPUT #3,AuditRecord$
      LPRINT TAB(4);AuditRecord$
   Wend
   CLOSE #3
   LPRINT "================================="
   LPRINT TAB(18);EXP_TOTAL#;TAB(26);ACC_TOTAL#
   RUN% = 1
   KILL "AUDIT.TXT"
RETURN

Check_Inventory:
   AbortRecipe% = 0 : Tracker% = 0
   RecipeID$ = RecipeHead$ + "00"
```

```
   InLoop% = 1
   While InLoop% = 1
      FetchFrom Recipes
      IfTableError 6 Then
         REM ' End of Recipe. Stop checking inventory and Materials
         InLoop% = 0
      ELSE

         FetchFrom Materials
         IfTableError 6 Then
            REM ' Material doesn't exist. Abort Recipe.
            Message$ = "-- "+MaterialID$+" --"
            Print "--No Such Mat.--" : Sleep 1000
            Print Message$ : Sleep 1000
            InLoop% = 0 : AbortRecipe% = 1
         ElseIf Inventory# < Amount# Then
            REM ' Not enough material.  Abort Recipe.
            Message$ = "-- "+MaterialID$+" --"
            Print "-Low Inventory-" : Sleep 1000
            Print Message$ : Sleep 1000
            InLoop% = 0 : AbortRecipe% = 1
         ENDIF
         GOSUB Get_Next_Ing
      ENDIF
   Wend
RETURN

Adjust_Inventory:
   FetchFrom Materials
   Inventory# = Inventory# - Delivered#
   StoreTo Materials
RETURN

Reports:
   MENUSEL$ = "Material_Report"
   WHILE RUN% = 1
      INPUT "^^Material_Report,Recipe_Report,Exit_Reports";MENUSEL$
      IF MENUSEL$ = "Material_Report" Then Gosub Material_Report
      IF MENUSEL$ = "Recipe_Report" Then Gosub Recipe_Report
      IF MENUSEL$ = "Exit_Reports" THEN RUN% = 0
   WEND
   RUN% = 1
   MENUSEL$ = "Reports"
RETURN

Material_Report:
   REM ' Prints the names and inventories of all materials in material file
   Print "Printing..."
   LPRINT "============================="
   LPRINT "|";TAB(6);"Materials Report";TAB(30);"|"
   LPRINT "|";TAB(4);DATE$;TAB(18);TIME$;TAB(30);"|"
   LPRINT "============================="
   LPRINT "|";TAB(5);"ID";TAB(15);"Inventory";TAB(30);"|"
   LPRINT "============================="
   OPEN MATERIALFILE$ FOR RANDOM AS #1 LEN = 18
   FIELD #1,10 as MaterialIDz$,8 as Inventory$
   INDEXED #1,MaterialIDz$
   i%=1
```

```
While EOF(1) = 0
      GET #1,i%
      i%=i%+1
      Inventory#=cvd(Inventory$)
      LPRINT TAB(5);MaterialIDz$;TAB(16);Inventory#
   Wend
   LPRINT "============================="
   CLOSE #1
RETURN

Recipe_Report:
   REM ' Prints the composition of all recipes in the recipe file
   Print "Printing..."
   LPRINT "================================================"
   LPRINT "I";TAB(6);"Recipe Report";TAB(22);DATE$;TAB(33);TIME$;TAB(50);"I"
   LPRINT "================================================"
   LPRINT "I";TAB(3);"Recipe";TAB(14);"Material
ID";TAB(28);"Scale";TAB(35);"Amount";TAB(46);"Tol.I"
   LPRINT "================================================"
   OPEN RECIPEFILE$ FOR RANDOM AS #2 LEN = 39
   FIELD #2,12 as RecipeIDz$,10 as MaterialIDz$,1 as ScaleIDz$,8 as Amount$,8 as
Tolerance$
   INDEXED #2,RecipeIDz$
   i%=1
   While EOF(2) = 0
      Get #2,i%
      KeyL% = INSTR(RecipeIDz$,"00") - 1
      CurrHead$ = LEFT$(RecipeIDz$,KeyL%)
      LPrint "   ";CurrHead$;":"
      ThisRecipe% = 1
      While EOF(2) = 0 AND ThisRecipe% = 1
         Get #2,i%
         Amount#=cvd(Amount$):Tolerance#=cvd(Tolerance$)
         ThisHead$ = LEFT$(RecipeIDz$,KeyL%)
         IF ThisHead$ = CurrHead$ Then
            LPRINT
TAB(14);MaterialIDz$;TAB(30);ScaleIDz$;TAB(36);Amount#;TAB(47);Tolerance#
            i%=i%+1
         ELSE
            ThisRecipe%=0
         ENDIF
      Wend
   Wend
   LPRINT "================================================"
   Close #2
RETURN

Get_Next_Ing:
   REM ' Determines value of RecipeID$, the next ingredient in a recipe
   Tracker% = Tracker% + 1
   Temp$ = STR$(Tracker%)
   Temp$ = LTRIM$(Temp$)
   IF LEN(Temp$)<2 THEN Temp$ = "0"+Temp$
   RecipeID$ = RecipeHead$+Temp$
Return

Get_Material_Info:
   REM ' This routine collects information about an ingredient in a recipe.
   REM ' This information includes Name, Scale, Amount and Tolerance.
   Temp% = Done%
   Menu$ = "^Mat.= ^"+MaterialString$
```

```
    INPUT Menu$;MaterialID$
    INPUT "^Scale ^A,B";ScaleID$
    Done% = 0
    While Done%=0
      INPUT "^Amount:^######.##",Amount#
      If Amount# = 0 Then
        Print "-Invalid Amount-" : Sleep 1000
      ELSE
        Done%=1
      Endif
    Wend
    Done%=0
    While Done%=0
      INPUT "^Tolerance:^####.##",Tolerance#
      If Tolerance# = 0 Then
        Print "--Invalid Tol.--" : Sleep 1000
      ELSE
        Done%=1
      Endif
    Wend
    Done% = Temp%
  RETURN

  Fill_Material_Array:
    REM ' Creates a list of materials seperated by commas in MaterialString$
    REM ' This  list is used to provide users with a list to choose from
    OPEN MATERIALFILE$ FOR RANDOM AS #1 LEN = 18
    FIELD #1,10 as MaterialIDz$,8 as Inventory$
    i%=1
    MaterialCount% = 0 : MaterialString$ =""
    WHILE EOF(1)=0
      GET #1
      TempString$ = RTRIM$(MaterialIDz$)
      IF TempString$ <> "" Then
        MaterialString$ = MaterialString$ + TempString$ +","
        i% = i%+1
        MaterialCount% = MaterialCount% + 1
      EndIf
    WEND
    CLOSE #1
    IF MaterialString$ = "" THEN
      MaterialString$ = "-EMPTY-"
    ELSE
      REM 'remove extra comma
      i% =  LEN(MaterialString$) - 1
      MaterialString$ = Left$(MaterialString$,i%)
    ENDIF
  RETURN

  Fill_Recipe_Array:
    REM ' Creates a list of recipes seperated by commas in RecipeString$
    REM ' This  list is used to provide users with a list to choose from
    OPEN RECIPEFILE$ FOR RANDOM AS #2 LEN = 39
    field #2,12 as RecipeIDz$,10 as MaterialIDz$,1 as ScaleIDz$,8 as Amount$,8 as
Tolerance$
    RecipeCount% = 0 : RecipeString$ =""
    WHILE EOF(2) = 0
      GET #2
      TempString$ = RTRIM$(RecipeIDz$)
      IF TempString$ <> "" Then
        IF RIGHT$(TempString$,2) = "00" THEN
```

```
                size% = LEN(TempString$) - 2
                TempString$ = LEFT$(TempString$,size%)
                RecipeString$ = RecipeString$ + TempString$ +","
                RecipeCount% = RecipeCount% + 1
            Endif
        EndIf
    WEND
    CLOSE #2
        i% =  LEN(RecipeString$)
        If i% = 0 THEN
            RecipeString$ = "-EMPTY-"
        ELSE
            i% = i%-1
            RecipeString$ = Left$(RecipeString$,i%) 'remove extra comma
        ENDIF
RETURN
```

Parts Counting

This example demonstrates how you can access the fine gross, net, and tare weights in shared data. This application is particularly useful for parts counting. It gives the highest internal resolution of the weights in double floating point format. The applicable fields are:

- Fine Gross Weight /wt117
- Fine Net Weight /wt118
- Fine Tare Weight /ws104

The following code executes the parts counting application.

```
5  DEFSHR DiscreteIn,p_100
10 DEFSHR TareA,t_690
20 DEFSHR TareAerr,s_290
30 DEFSHR DiscreteOut,p_503
40 DEFSHR ClearTareA,t_691
50 DEFSHR MotionA,s_200
55 DEFSHR NetWt,wt118
56 DEFSHR GrossWt,wt117
57 DEFSHR TareWt,ws104
60 PRINT "Place Container"
70 IF DiscreteIn=0 THEN GOTO 70
75 PRINT "Taring Container"
80 SLEEP 3000
90 TareA=1
100 IF TareA=1 THEN GOTO 100
110 IF TareAerr=0 THEN GOTO 160
120 PRINT "Tare Failed"
130 SLEEP 1000
150 GOTO 90
160 PRINT "Place 10 Parts"
170 IF DiscreteIn=0 THEN GOTO 170
180 PRINT "Weighing Sample"
190 IF MotionA=1 THEN GOTO 190
200 sampleWt#=NetWt/10.0
205 LPRINT "gross weight=";GrossWt;"  tare weight=";TareWt
206 LPRINT "net weight=";NetWt;"  piece weight=";sampleWt#
```

```
210 SLEEP 1000
220 PRINT "Place All Parts"
230 IF DiscreteIn=0 THEN GOTO 230
240 PRINT "Weighing Parts"
250 IF MotionA=1 THEN GOTO 250
260 parts%=cint(NetWt/sampleWt#)
265 LPRINT "total parts weight=";NetWt;"  number parts=";parts%
266 LPRINT ""
270 SLEEP 1000
280 PRINT "Num Parts=";parts%
290 SLEEP 3000
300 IF NetWt > 0.0 THEN GOTO 300
310 PRINT "Completed"
320 SLEEP 3000
330 ClearTareA=1
340 IF ClearTareA=1 GOTO 340
350 GOTO 60
```

Printer Templates

You can read and write printer templates from JagBASIC. This sample program demonstrates reading templates from JAGXTREME Shared Data and saving them in a sequential file.

```
1 REM This is a sample program for reading templates from JagBASIC
2 REM and saving it in a files called templat1.dat thru templat5.dat.
3 REM Dimension an array of strings large enough to hold the template,
4 REM and "DEFSHR" the first element of the array to the template.
5 REM The maximum string size in JagBASIC is 80 bytes.
6 REM The maximum template size is 400 bytes.
7 REM Reading of shared data is done when you access the first
8 REM element, so read the first element first.
100 DIM go%(5),T$(6)
120 go%(1)=300:go%(2)=400:go%(3)=500:go%(4)=600:go%(5)=700
130 INPUT "^Save Template? ^1,2,3,4,5",c$
140 i%=asc(c$)-48
150 switchsub go%(i%)
160 FOR i%=1 to 6
170 IF len(T$(i%))<>0 THEN write #1,T$(i%)
180 NEXT i%
190 CLOSE #1
200 END
300 OPEN "TEMPLAT1.DAT" FOR OUTPUT AS#1
310 DEFSHR T$(1),PTP01
320 RETURN
400 OPEN "TEMPLAT2.DAT" FOR OUTPUT AS#1
410 DEFSHR T$(1),PTP02
420 RETURN
500 OPEN "TEMPLAT3.DAT" FOR OUTPUT AS#1
510 DEFSHR T$(1),PTP03
520 RETURN
600 OPEN "TEMPLAT4.DAT" FOR OUTPUT AS#1
610 DEFSHR T$(1),PTP04
620 RETURN
700 open "TEMPLAT5.DAT" FOR OUTPUT AS#1
710 DEFSHR T$(1),PTP05
720 return
```

In the JAGXTREME terminal, you can read and write printer templates from JagBASIC. This sample program demonstrates loading printer templates from a sequential file and writing them into JAGXTREME Shared Data.

```
1 REM This is a sample program for writing printer templates
2 REM that are saved a files called templat1.dat thru tempat6.dat.
3 REM Dimension an array of strings large enough to hold the template,
4 REM and "DEFSHR" the first element of the array to the template.
5 REM The maximum string size in JagBASIC is 80 bytes.
6 REM The maximum template size is 400 bytes.
7 REM Writing of shared data templates is done when you access the first
8 REM element, so write the first element last.
100 DIM go%(5),T$(6),buf$(6)
120 go%(1)=300:go%(2)=400:go%(3)=500:go%(4)=600:go%(5)=700
130 INPUT "^Load Template? ^1,2,3,4,5",c$
140 i%=ASCc(c$)-48
150 SWITCHSUB go%(i%)
```

```
160 FOR i%=1 to 6
170 IF NOT EOF(1) THEN INPUT#1,BUF$(i%):last%=i%
180 NEXT i%
190 FOR i%=last% to 1 step -1
200 T$(i%)=buf$(i%)
210 NEXT i%
220 CLOSE #1
230 END
300 OPEN "TEMPLAT1.DAT" FOR INPUT AS #1
310 DEFSHR T$(1),PTP01
320 RETURN
400 open "TEMPLAT2.DAT" FOR INPUT AS #1
410 DEFSHR T$(1),PTP02
420 RETURN
500 OPEN "TEMPLAT3.DAT" FOR INPUT AS #1
510 DEFSHR T$(1),PTP03
520 RETURN
600 OPEN "TEMPLAT4.DAT" FOR INPUT AS #1
610 DEFSHR T$(1),PTP04
620 RETURN
700 OPEN "TEMPLAT5.DAT" FOR INPUT AS #1
710 DEFSHR T$(1),PTP05
720 RETURN
```

You can read and write printer templates from JagBASIC. This sample program demonstrates creating a printer templates in JagBASIC and writing it to JAGXTREME Shared Data.

```
1 REM This is a sample program for creating a printer template.
2 REM
10 REM These are some template format samples:
11 REM
12 REM /D=40
13 REM     |    |
14 REM     |    +--> Repeat Occurrences
15 REM     +-----> Character Value
16 REM
17 REM  will print... ========================================
18 REM
19 REM     /n3 will print... three LF/CR characters.
20 REM
21 REM /E0 signifies the end of the template.
20 REM
22 REM /jag19!/L15/!
23 REM \_____/ | |
24 REM         |    | |
25 REM      |    | +-> Max Length
26 REM      |    |
27 REM      |    +--> Justify (R)ight
28 REM      |           (L)eft or (C)enter
29 REM      |
30 REM      +---------> Field Path Name
31 REM
32 REM  /wt101          will print...
33 REM  /wt101 field in default format, left justified, default length.
34 REM
```

```
35 REM  /wt201/R     will print...
36 REM  /wt201 field right justified, default length.
37 REM
38 REM  /wt202/C040  will print...
39 REM  /wt202 field centered in a 40 byte area.
40 REM

100 DIM y$(5)
120 DEFSHR y$(1),ptp04
130 y$(3)="/wt103 !/ws109!/n1/Net Weight:     !/wt102 !/wt103!/n3/!/E0"
140 y$(2)="Gross Weight:   !/wt101 !/wt103!/n1/Tare Weight:    !/ws102 !"
150 y$(1)="!/jag19!/L15/!/jag20!/L15/!/cs118!/R10/!/n1/!/D=40/!/n1/"
160 LPRINT "done"
```

JOG Example

This is a program for using Jog Setpoints. Jog setpoints are based on time rather than weight. They are typically used when the flow of material is very fast compared to the amount of material that needs to be weighed. For example, they can be used at the end of an order to add a small amount of material to bring an order into its weight tolerance.

```
REM ************************************
REM                  DEFSHR's
REM ************************************
REM Define the jog table.
REM You can have up to 10 jog weights and corresponding
REM jog times in the Jog table.  The jog setpoint
REM interpolates between the next higher and next lower
REM jog weight to determine a specific jog time.
REM The Jog Table is in Shared Data Variables clv01-clv20.
REM The values are floating point, stored in string format.
REM The Jog Weights are in clv01-clv10 in ascending order.
REM You can prematurely end the table with a "0" entry.
REM The corresponding jog times are in clv11-clv20.
DIM jogWt$(10)
DIM jogTm$(10)
DEFSHR jogWt$(1),clv01
DEFSHR jogWt$(2),clv02
DEFSHR jogWt$(3),clv03
DEFSHR jogWt$(4),clv04
DEFSHR jogWt$(5),clv05
DEFSHR jogWt$(6),clv06
DEFSHR jogWt$(7),clv07
DEFSHR jogWt$(8),clv08
DEFSHR jogWt$(9),clv09
DEFSHR jogWt$(10),clv10
DEFSHR jogTm$(1),clv11
DEFSHR jogTm$(2),clv12
DEFSHR jogTm$(3),clv13
DEFSHR jogTm$(4),clv14
DEFSHR jogTm$(5),clv15
DEFSHR jogTm$(6),clv16
DEFSHR jogTm$(7),clv17
DEFSHR jogTm$(8),clv18
DEFSHR jogTm$(9),clv19
DEFSHR jogTm$(10),clv20
```

```
REM gate discrete inputs
DEFSHR FillGateOpened%,p_100
DEFSHR DischargeOpened%,p_103

REM discrete outputs to gates
DEFSHR OpenFill%,p_501
DEFSHR OpenDischarge%,p_503

REM scale DEFSHR's
DEFSHR ScaleWeight#,wt110
DEFSHR ScaleMotion%,s_200

REM jog setpoint DEFSHR's
DEFSHR spen%,sp102
DEFSHR sptar%,sp103
DEFSHR coin#,sp105
DEFSHR latch%,sp188
DEFSHR setsp%,t_698

REM ***********************************
REM                    Initialization Logic
REM ***********************************
REM close the gates
spen%=0:setsp%=1
OpenFill%=0:OpenDischarge%=0

REM initialize ladder logic
REM "t_61c" starts the setpoint jog timer.
REM Move the "fill gate opened" input to "t_61c".
REM Move the "setpoint feeding output" to "open fill gate".
NEWLADDER
RUNGMOV p_100,t_61c
RUNGMOV s_210,p_500

REM ***********************************
REM          Main Menu
REM ***********************************
MainMenu:
m$="Learn"
input "^Menu^ Learn,Jog,Exit",m$
IF m$="Learn" THEN GOSUB LearnMode
IF m$="Jog" THEN GOSUB JogMode
IF m$="Exit" THEN End
GOTO MainMenu

REM ***********************************
REM Setting up a Learn Setpoint
REM ***********************************
REM Set the jog time in the coincidence value for the setpoint.
REM You can determine the weight associated with the jog weight
REM by reading the gross weight before and after the setpoint.
REM The Learn setpoint is latched so you need to
REM reset the latch before starting the setpoint.
LearnSetpoint:
spen%=1
```

```
sptar%="L"
coin#=JogTime#
latch%=0
setsp%=1
RETRUN

REM ***********************************
REM Setting up a Jog Setpoint
REM ***********************************
REM Set the jog weight in the coincidence value for the setpoint.
REM The jog setpoint logic uses the Jog Tables to determine
REM the amount of time to hold its feeding output open.
REM The Jog setpoint is latched so you need to
REM reset the latch before starting the setpoint.
JogSetpoint:
spen%=1
sptar%="J"
coin#=JogWeight#
latch%=0
setsp%=1
return

REM ***********************************
REM   Learn Mode Logic
REM ***********************************
LearnMode:
MinJogTime%=100
input "^Min Jog ms.^####";MinJogTime%
MaxJogTime%=3000
input "^Max Jog ms.^####";MaxJogTime%

REM set jog table times
TimeIncrement%=(MaxJogTime%-MinJogTime%)/9
jogTm$(1)="100":jogTm$(10)=str$(MaxJogTime%)
for TablePos%=2 to 9
jogTm$(TablePos%)=str$(val(jogTm$(TablePos%-1))+TimeIncrement%)
next TablePos%

REM build jog table weights
for TablePos%=1 to 10
CheckReady:
m$="Yes"
n$="^Jog "+str$(TablePos%)+"^ Yes,No,Exit"
input n$;m$
IF m$="No" THEN GOTO CheckReady
IF m$="Exit" THEN End
GOSUB WaitFillGateClosed:GOSUB WaitDischargeClosed:GOSUB WaitMotion
TareWeight#=ScaleWeight#
JogTime#=val(jogTm$(TablePos%))
GOSUB LearnSetpoint

REM wait until setpoint logic opens then closes fill gate
GOSUB WaitFillGateOpened:GOSUB WaitFillGateClosed
REM wait for scale motion to settle
print "settling"
SLEEP 5000:GOSUB WaitMotion
```

```
REM set jog table weights
LearnedWt#=ScaleWeight#-TareWeight#
GOSUB CheckDischargeScale
jogWt$(TablePos%)=str$(LearnedWt#)
next TablePos%


REM print jog table
FOR TablePos%% = 1 to 10
LPRINT jogTm$(TablePos%),jogWt$(TablePos%)
NEXTTablePos%%

GOTO MainMenu


REM ***********************************
REM   Jog Mode Logic
REM ***********************************
JogMode:
JogWeight#=0
input "^Weight^#####";JogWeight#
GOSUB WaitFillGateClosed:GOSUB WaitDischargeClosed:GOSUB WaitMotion
TareWeight#=ScaleWeight#
GOSUB JogSetpoint

REM wait until setpoint logic opens then closes fill gate
GOSUB WaitFillGateOpened:GOSUB WaitFillGateClosed
REM wait for scale motion to settle
print "settling"
SLEEP 5000:GOSUB WaitMotion

print "Wt =" + str$(ScaleWeight#-TareWeight#)
WaitJogModeKey:
m$=inkey$:IF m$="" THEN GOTO WaitJogModeKey
GOSUB CheckDischargeScale
return

REM ***********************************
REM          Gate Open/Close Routines
REM ***********************************
WaitFillGateOpened:
IF FillGateOpened%=0 THEN GOTO WaitFillGateOpened
return

WaitFillGateClosed:
IF FillGateOpened%=1 THEN print "Jogging":GOTO WaitFillGateClosed
return

CheckDischargeScale:
m$="Yes"
input "^Discharge^ Yes,No";m$
IF m$="No" THEN return
OpenDischarge%=1
print "Discharging"
WaitScaleEmpty:
IF ScaleWeight#>20.0 THEN GOTO WaitScaleEmpty
OpenDischarge%=0
```

```
WaitDischargeClosed:
IF DischargeOpened%=1 THEN print "Closing Discharge":GOTO WaitDischargeClosed
return

REM ***********************************
REM          Motion Routine
REM ***********************************
WaitMotion:
IF ScaleMotion%=1 THEN print "Motion":GOTO WaitMotion
RETURN
```

JagBASIC SMTP Client Program

The example program below illustrates how a JAGXTREME terminal can send email using Simple Mail Transfer Protocol (SMTP). In this example, the terminal acts as the client. It establishes a connection with the SMTP server, sends an email message, terminates the connection with the server, and then loops back to the beginning of the program. A server utilizing SMTP must be present on the network for email to be sent and received. The IP address on the third line below "CONN1:" must be replaced with the IP address of the machine on which the SMTP server resides. The JagBASIC preprocessor must be used to prepare the program to run on the terminal.

```
REM  Send Email using SMTP
crlf$ = CHR$(13) + CHR$(10)
lf$ = CHR$(10)
cr$ = CHR$(13)

REM     Establish Connection
CONN2:
   sock%=socket()
   IF sock%=0 THEN print "No Socket Free":SLEEP 2000: GOTO CONN2
CONN1:
   stat% = sockopt(sock%,-1)
   REM connect must be called with server's IP address
   stat% = connect(sock%,"146.207.104.023",25)
   IF stat% = -1 THEN print "connecting":sleep200:GOTO CONN1
   IF stat% =  0 THEN print "trying connect":stat%=sockcls(sock%):SLEEP
        2000:GOTO CONN2
   PRINT "connect success":SLEEP 200

REM     Wait for Service Ready (220)
   begin_time# = TIMER()
   cmd%=0
   WHILE cmd% <> 1 and TIMER()-begin_time# < 2.0
      rcvstr$=RECV$(sock%,160)
      GOSUB INTERP_CMD
   WEND
   IF cmd% <> 1 THEN GOTO TIME_EXP
   PRINT "Service Ready"


REM     Send HELO
```

```
   sndstr$ = "HELO terminal"+crlf$:len%=len(sndstr$)
   stat%=send(sock%,sndstr$)
   IF stat%<>len% THEN PRINT "send failed": SLEEP 2000: GOTO ENDPRG

REM     Wait for OK
   GOSUB WAIT_FOR_OK

REM     Send MAIL From:.......
   sndstr$ = "MAIL From:<jaguar@mt.com>"+crlf$:len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

REM     Wait for OK
   GOSUB WAIT_FOR_OK

REM     Send RCPT To:.............
   sndstr$ = "RCPT To:<ScaleAdmin@mt.com>"+crlf$:len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

REM     Wait for OK
   GOSUB WAIT_FOR_OK

REM     Send DATA
   sndstr$ = "DATA"+crlf$:len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

REM     Wait for Start Mail Input (354)
   cmd%=0
   WHILE cmd% <> 3
      rcvstr$=RECV$(sock%,160)
      GOSUB INTERP_CMD
   WEND
   PRINT "Begin Mail trans"

REM     Send Message Body
   sndstr$ = "Subject: Automated Email Alert!"+crlf$
   sndstr$ = sndstr$ + "Mettler Toledo JAGXTREME"+crlf$
   len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

   sndstr$ = "This is a test."+crlf$+"Hello World!"+crlf$
   len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

REM     Send <crlf>.<crlf> to terminate message.
   sndstr$ = crlf$+"."+crlf$
   len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

REM     Wait for OK
   GOSUB WAIT_FOR_OK

REM     Send QUIT to terminate connection
   sndstr$ = "QUIT"+crlf$:len%=len(sndstr$)
   stat%=send(sock%,sndstr$)

REM     Wait for Goodbye (221)
```

```
    cmd%=0
    WHILE cmd% <> 4
       rcvstr$=RECV$(sock%,160)
       GOSUB INTERP_CMD
    WEND
    PRINT "Goodbye received":SLEEP 500
    GOTO ENDPRG

INTERP_CMD:
    cmd%=0
    a$ = LEFT$(rcvstr$,3)
    IF a$="220" THEN cmd% = 1 : REM Service Ready
    IF a$="250" THEN cmd% = 2 : REM OK
    IF a$="354" THEN cmd% = 3 : REM Start Mail Input
    IF a$="221" THEN cmd% = 4 : REM Closing Connection
    IF a$ = CHR$(13) THEN cmd% = 8
    IF a$ = CHR$(10) THEN cmd% = 9
    return

WAIT_FOR_OK:
  RETRY:
    rcvstr$=RECV$(sock%,160)
    IF rcvstr$="" THEN GOTO RETRY
    GOSUB INTERP_CMD
    IF cmd%<>2 THEN PRINT "No OK received";cmd%:SLEEP 2000:print rcvstr$
    IF cmd%<>2 THEN SLEEP 2000:GOTO ENDPRG
    return

TIME_EXP:
    print "connect. expired"
    SLEEP 500

ENDPRG:
    print "closing socket"
    SLEEP 500
    stat%=sockcls(sock%)
    SLEEP 10000 : REM wait 10 seconds before looping back
    GOTO CONN2  : REM loop back and send message again

END
```

VISUAL BASIC SMTP Server Program

An SMTP server program can be written if one does not exist. The sample program below is written in Microsoft Visual Basic 6.0 and illustrates a simple server program. This server program can receive email messages from devices using SMTP and store them as text files in the subdirectory "\mail" off the installation directory. The messages are stored with the default name "MAIL###.TXT", where the #'s are replaced with numbers. The contents of a message can be viewed by clicking its name in the file list box. The selected message can be printed to the default printer or deleted from the directory by clicking the "Print" or "Delete" buttons respectively. Following is a listing of the source code and a property table that can be used to generate the sample SMTP server application.

```
Private intMax As Long
Dim PATH As String
Const MAXMSG = 150
Dim PathandName As String
Dim messageTxt(500) As String
Dim CRLF As String


Function FileExists(p As String) As Long
  If Dir(p) <> "" Then
    FileExists = 1    ' Return 1 indicating file exists.
  Else
    FileExists = 0    ' Return 0 indicating file does not exist.
  End If
End Function

Private Sub cmdDelete_Click()
   Dim Message As String
   Dim ButtonsAndIcons As Integer
   Dim Title As String
   Dim Response As Integer

   'Delete message if one has been selected
   If PathandName <> "" Then
      Message = "Delete EMail Message?"
      ButtonsAndIcons = vbYesNo + vbQuestion
      Title = "Delete Message?"

      Response = MsgBox(Message, ButtonsAndIcons, Title)

      If Response = vbYes Then
         FileThere = FileExists(PathandName) 'Check that file exists
         If FileThere Then
            Kill PathandName           'Delete the file
            txtMailText.Text = ""      'clear text in message box
            PathandName = ""
         End If
         flstFileBox.Refresh        'update the file list box
      End If

   End If
End Sub

Private Sub cmdExit_Click()
   Beep
   End
End Sub

Private Sub cmdPrint_Click()
   'Print the text box's contents to the default printer.
   Printer.Font.Name = "Courier"
   Printer.Font.Size = 12
   Printer.Print txtMailText.Text
   Printer.EndDoc
End Sub
```

```
Private Sub flstFileBox_Click()
' When the user clicks on a message displayed in the file list box,
' display the text of that message on the screen.
   If flstFileBox.FileName <> "" Then
      PathandName = PATH + flstFileBox.FileName
      FileThere = FileExists(PathandName)     'Check if file exists

      If FileThere Then
         FileNum = FreeFile

                  'get date and time file was received/modified
         FileLastModified = CStr(FileDateTime(PathandName))
         Open PathandName For Input As FileNum
   txtMailText.Text = "   --" + flstFileBox.FileName + " "
   txtMailText.Text =txtMailText.Text + FileLastModified + "--" + CRLF
   txtMailText.Text = txtMailText.Text + Input(LOF(FileNum), FileNum)
         Close FileNum
      Else
         'The file did not exist.
         'Update the list of files and clear the text box.
         flstFileBox.Refresh
         txtMailText.Text = ""
      End If
   End If
End Sub

Private Sub Form_Load()
   On Error Resume Next
   MkDir "MAIL"
   PATH = CurDir + "\MAIL\"

   flstFileBox.PATH = PATH
   PathandName = ""
   intMax = 0
   tcpServer(0).LocalPort = 25
   tcpServer(0).Listen
   CRLF = Chr(13) + Chr(10)
End Sub

Private Sub tcpServer_Close(Index As Integer)
   tcpServer(Index).Close
End Sub

Private Sub tcpServer_ConnectionRequest _
(Index As Integer, ByVal requestID As Long)
  If Index = 0 And flstFileBox.ListCount < MAXMSG Then
    intMax = intMax + 1
    Load tcpServer(intMax)
    tcpServer(intMax).LocalPort = 25
    tcpServer(intMax).Accept requestID
    tcpServer(intMax).SendData "220 s-jcr1.sjcr1.com" + CRLF
  End If
End Sub

Private Sub tcpServer_DataArrival(Index As Integer, ByVal bytesTotal As Long)
'Look for SMTP commands.  If one is present, begin dialog and save
'received information.
```

```
Dim strData As String
   tcpServer(Index).GetData strData
   tmpstr = Left(strData, 4)
   Length = Len(strData)

   Select Case tmpstr
   Case "HELO"
      messageTxt(Index)="SENDER DOMAIN:  "+ Mid(strData,6,Length - 5)
      tcpServer(Index).SendData "250 s-jcr1.sjcr1.com" + CRLF
   Case "MAIL"
      messageTxt(Index) = messageTxt(Index) + "FROM:  " + Mid(strData, 11,
 Length - 12) + " " + tcpServer(Index).RemoteHostIP + CRLF
      strSend = "250 OK" + CRLF
      tcpServer(Index).SendData (strSend)
   Case "RCPT"
   messageTxt(Index)=messageTxt(Index)+"TO:  "+Mid(strData,9,Length-8)
      strSend = "250 OK" + CRLF
      tcpServer(Index).SendData (strSend)
   Case "DATA"
      strSend = "354 Start mail input" + CRLF
      tcpServer(Index).SendData (strSend)
   Case "QUIT"
      strSend = "221 Closing Connection" + CRLF
      tcpServer(Index).SendData (strSend)
      If flstFileBox.ListCount = MAXMSG Then
         WMessage ="Mailbox full. Delete old messages to make room."
         ButtonsAndIcons = vbOKOnly
         Beep
         junk = MsgBox(WMessage, ButtonsAndIcons, "Mailbox Full!")
      End If
   Case Else
      position = InStr(strData, CRLF + "." + CRLF)
      If position <> 0 Then
        messageTxt(Index)=messageTxt(Index)+Mid(strData,1,position-1)
         strSend = "250 OK" + CRLF
         tcpServer(Index).SendData (strSend)
         strSmall = Mid(strData, 1, position - 1)

         'Received complete message, search for next available
         'file name and save message.
         msgcounter = 0: foundname = 0
         Do While msgcounter < MAXMSG
            msgcounter = msgcounter + 1
            numstr = CStr(msgcounter)
            numstr = Format(numstr, "000")
            mailname = "MAIL" + numstr + ".TXT"
            inlist = 0: countit = 0
            Do While countit < flstFileBox.ListCount
               If mailname = flstFileBox.List(countit) Then
                  inlist = 1
                  Exit Do
               End If
               countit = countit + 1
            Loop
            If inlist = 0 Then
               foundname = 1
```

```
                Exit Do
            End If
        Loop
        If foundname = 1 Then
            mailname = PATH + mailname
            FileNum = FreeFile
            Open mailname For Output As FileNum
            Write #FileNum, messageTxt(Index)
            Close FileNum
        End If
        flstFileBox.Refresh
    Else
        messageTxt(Index) = messageTxt(Index) + strData
    End If
  End Select
End Sub
```

Object	Property	Setting
Form	**Name**	**frmSample**
	Caption	Sample SMTP Server
Command Button	**Name**	**cmdDelete**
	Caption	D&elete
Command Button	**Name**	**cmdPrint**
	Caption	&Print
Command Button	**Name**	**cmdExit**
	Caption	E&xit
File List Box	**Name**	**flstFileBox**
	Pattern	*.txt
Winsock	**Name**	**tcpServer**
	Protocol	sckTCPProt
	Index	0
Text Box	**Name**	**txtMailText**
	ScrollBars	Both
	Text	(empty)
	MultiLine	True

9 Error Codes and Messages

This section discusses error messages that may be output to the LPRINT device during debugging or program execution. The JAGXTREME terminal lower display will show the Error Number Code and Line Number, with the error message being output to the LPRINT device (a printer or a PC running a communication or terminal emulation program). For example, the error *Unknown Command* would show up on the JAGXTREME terminal display as: *E26 L 1010*. The message output to the LPRINT device should show as:

ERROR in line 0: Unknown command.

Common Errors

Some common errors and troubleshooting tips are as follows:

- For Upload/Download problems, set the JAGXTREME terminal in Diagnostic Test mode. This tests the transmit and receive lines from the PC to the JAGXTREME terminal.
- If a file downloads OK to the JAGXTREME terminal, but will not load (E2L0 error), check for blank lines and no line numbers.

Error Codes

The following is a listing of possible error codes and messages in JagBASIC

Error Code	Error Message	Description	Problem Cause	Remedy
0	File open failed	JagBASIC programming error.	JagBASIC attempted to open a nonexistent RAMDISK file or serial communications device.	Correct the JagBASIC program.
1	Memory find fail	JagBASIC programming error.	JagBASIC exceeded the memory limits of the system.	Reduce lines. Eliminate unnecessary spaces in program. Reduce variables. Reduce size of arrays. When chaining JagBASIC programs, chain in the largest program first to reduce memory fragmentation.
2	Line # invalid	JagBASIC programming error.	JagBASIC contains a line number greater than 30000 or is a duplicate of an existing line number.	Correct the JagBASIC program.

Error Code	Error Message	Description	Problem Cause	Remedy
3	Resource in use	JagBASIC programming error.	JagBASIC tried to access a system resource in use by another JAGXTREME terminal task. JagBASIC cannot open a serial port that has been assigned to a serial port connection in setup. When two or more JagBASIC applications share a remote serial port, only one can have the port open at a time.	Correct JagBASIC application. To share remote serial ports between multiple JagBASIC applications, develop sharing logic that checks for this specific error code.
4	LOAD:no filename	Operator error.	The LOAD command does not contain a file name.	Correct the command.
5	No line number	JagBASIC programming error.	The program line does not have a line number.	Correct the JagBASIC program.
6	Record not found	JagBASIC programming error.	A record specified in a GET statement for an indexed sequential file could not be found in the file.	There should be an ON ERROR statement in the JagBASIC program to handle these potential situations.
7	RETURN no GOSUB	JagBASIC programming error.	RETURN statement is present without required GOSUB.	Correct the JagBASIC program.
8	Incomplete line	JagBASIC programming error.	JagBASIC program contains a line that does not have the full syntax required for a line.	Correct the JagBASIC program.
9	ON no GOSUB	JagBASIC programming error.	ON statement is present without required GOSUB.	Correct the JagBASIC program.
10	Value out range	JagBASIC programming error.	The JagBASIC statement is referring to a value out of the range of acceptable values.	Correct the JagBASIC program.
11	Syntax error	JagBASIC programming error.	The JagBASIC program has a syntax error.	Correct the JagBASIC program.
12	Invalid device #	JagBASIC programming error.	The JagBASIC program is referencing a device # that is not open.	Correct the JagBASIC program.
13	Device error	JagBASIC programming error.	The JagBASIC program has referred to an illegal device or a device that is not open.	Correct JagBASIC program.
14	Command error	An error occurred in trying to access a file from the RAMDISK.	You tried to access a file that does not exist or the file system has been corrupted.	Use the DIR command from the JagBASIC Interpreter to verify the directory of the RAMDISK. If the file system has been corrupted, re-initialize it from the JagBASIC setup menus and rebuild it from the backup files you are maintaining on a PC.

Error Code	Error Message	Description	Problem Cause	Remedy
14	Command error	An error occurred in trying to access a file from the RAMDISK.	You tried to access a file that does not exist or the file system has been corrupted.	Use the DIR command from the JagBASIC Interpreter to verify the directory of the RAMDISK. If the file system has been corrupted, re-initialize it from the JagBASIC setup menus and rebuild it from the backup files you are maintaining on a PC.
15	Chain Context	JagBASIC programming error.	A chain statement inside a subroutine, for-next, while loop, or if statement.	Chain only from top level of JagBASIC program.
16	Event def error	JagBASIC programming error.	Programming error in defining an event.	Correct the JagBASIC program.
17	Type mismatch	JagBASIC programming error.	JagBASIC statement is using an invalid data type or is relating two incompatible data types.	Correct the JagBASIC program.
18	DIM not array	JagBASIC programming error.	JagBASIC program has attempted to dimension a variable that is not an array.	Correct the JagBASIC program.
19	Out of data	JagBASIC programming error.	JagBASIC program has issued more READ commands to initialize system variables than data specified in DATA statements.	Correct the JagBASIC program.
20	Overflow	JagBASIC programming error.	A JagBASIC program causes an overflow error by exceeding certain system limits. The maximum size of the GOSUB stack, the FOR-NEXT stack, and the WHILE-WEND stack is 9 entries each. If you try to nest subroutines more than 9 entries deep, you get an overflow error. Overflow errors can also be caused by syntax errors.	Correct the JagBASIC program.
21	NEXT without FOR	JagBASIC programming error.	There is a NEXT statement without the required FOR statement.	Correct the JagBASIC program.
22	Undefined funct.	JagBASIC programming error.	The JagBASIC statement is referring to an undefined function.	Correct the JagBASIC program.

Error Code	Error Message	Description	Problem Cause	Remedy
23	Divide by zero	JagBASIC programming error.	JagBASIC program attempted to divide a number by zero.	Correct the JagBASIC program.
24	Can't redim. var	JagBASIC programming error.	Once a JagBASIC application has declared a variable or an array, it cannot later be redimensioned to a different size array.	Correct JagBASIC program.
25	OPTION BASE->DIM	JagBASIC programming error.	The JagBASIC program must define the OPTION BASE before dimensioning an array.	Correct the JagBASIC program.
26	Illegal Command	JagBASIC programming error.	The JagBASIC program has issued a command that is not a legal command.	Correct the JagBASIC program.
27	Too many dimens.	JagBASIC programming error.	JagBASIC arrays can have at most three dimensions.	Correct the JagBASIC program.
28	Invalid SD name	JagBASIC programming error.	The JagBASIC program is referencing an invalid Shared Data name.	Correct the JagBASIC program.
29	Program too big	JagBASIC programming error.	The program exceeds 300 text lines or 15 KB. You are typing in a JagBASIC program at the JAGXTREME terminal when the temporary program buffer becomes full.	For the first problem, separate the program into smaller files that can be run independently or be chained together. When chaining, always start execution with the largest program to avoid memory fragmentation. For the second problem, save the current program and re-load it. This will cause a larger temporary program buffer to be allocated.
30	Line too big	JagBASIC programming error.	A JagBASIC line is greater than 80 characters.	Correct the JagBASIC program.
31	SD string > max.	JagBASIC programming error.	JagBASIC can only access shared data fields whose length is less than the maximum JagBASIC string size of 80 bytes.	Correct the JagBASIC program.
32	No Remote Access	JagBASIC programming error.	The program is attempting to access a device that is already in use by a serial connection or by another JagBASIC program in the JAGXTREME terminal cluster.	To access a serial device, you must remove all continuous output or input connections to the serial device in setup. To share a device among JagBASIC programs, you must setup a scheme where only one program has the device open at a time.

10 ASCII/HEX Code Chart

Char.	Dec.	Hex.
NUL	0	00
SOH	1	01
STX	2	02
ETX	3	03
EOT	4	04
ENQ	5	05
ACK	6	06
BEL	7	07
BS	8	08
HT	9	09
LF	10	0A
VT	11	0B
FF	12	0C
CR	13	0D
SO	14	0E
SI	15	0F
DLE	16	10
DC1	17	11
DC2	18	12
DC3	19	13
DC4	20	14
NAK	21	15
SYN	22	16
ETB	23	17
CAN	24	18
EM	25	19
SUB	26	1A
ESC	27	1B
FS	28	1C
GS	29	1D
RS	30	1E
US	31	1F

Char.	Dec.	Hex.
SP	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F

Char.	Dec.	Hex.
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
[	91	5B
\	92	5C
]	93	5D
^	94	5E
_	95	5F

Char.	Dec.	Hex.
`	96	60
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
|	124	7C
}	125	7D
~	126	7E
DEL	127	7F

NOTES

11 Appendix 1

JagBASIC Commands

This appendix provides a quick alphabetic reference to all JagBASIC commands.

Command	Usage	Page
ABS()	Returns the absolute value of a number.	5-30, 5-31
ACCEPT	Allows JagBASIC application to accept new connection requests that remote clients are initiating.	5-97, 5-99
AND	A logical operator in a decision statement which establishes two sets of criteria to be met.	5-20
ASC()	Returns the ASCII or extended code value for the first character in a string expression.	5-36, 5-37
ATN()	Returns the arctangent of a specified numeric expression in radians.	5-30, 5-31
BEEP	Sounds the beeper tone for the specified milliseconds.	5-45, 5-46
BREAK	Stops execution of program at line number	5-3
CHAIN	Dynamically loads another program file for execution and begins executing the program.	5-20, 5-21
CHAINCALL	Operates the same as a CHAIN command except that it remembers the current program name and line number of the program that is initiating the chaining.	5-20, 5-21
CHAINRET	Operates the same as a CHAIN command except that is returns control from the chained program to the chaining program at the next line after the CHAINCALL.	5-20, 5-22
CHR$()	Returns the single-character string corresponding to the specified ASCII code.	5-36, 5-37
CINT	Rounds a numeric expression to the closest integer.	5-30, 5-31
CKSUM$	Generates a checksum and returns as a string.	5-53
CLEAR	Closes all files, releases file buffers, clears all common variables, sets numeric variables and arrays to zero and sets string variables to null.	5-3, 5-4
CLKTICK	Returns a double float number that is the number of clock ticks that have occurred since the last power up of the terminal.	5-92
CLOSE	Closes a file or serial port.	5-53, 5-54, 5-65, 5-66
CLREVENT	Clears outstanding event triggers.	5-80, 5-82
COMBITS	Returns the state of the input modem signals on the COM3 serial port.	5-53, 5-54, 5-82, 5-86
COMMON	Defines global variables that can be shared between chained programs.	5-11, 5-12
CONNECT	Initiates a TCP/IP connection to a remote host.	5-97, 5-99

Command	Usage	Page
COS()	Returns the cosine of a specified angle expressed in radians.	5-30, 5-32
CRC$	Generates CRC and returns as a string.	5-53, 5-54
CSNG()	Converts a numeric expression to a single-precision value.	5-30, 5-32
CVI, CVS, CVD	Convert string variable types to numeric variable types.	5-65, 5-67
DATA	Specifies values to be read by READ statements.	5-11, 5-12
DATE$	Sets or returns the JAGXTREME system date.	5-92, 5-93
DEFSHR EVENT	Allocates a shared data event.	5-80, 5-83
DEFSHR	Allows a program to access the JAGXTREME terminal shared database.	5-11, 5-13
DELETE	Deletes a specific program line or a range of lines.	5-3, 5-4
DELEVENT	Deallocates an event.	5-80, 5-83
DELREC	Deletes a record from the indexed sequential file.	5-65, 5-67
DIM	Declares an array, where subscripts are the dimensions of the array.	5-11, 5-16
DIR	Displays the RAMDISK directory on the LPRINT device.	5-3, 5-4
DISABLE	Disables asynchronous event triggers.	5-80, 5-83
ENABLE	Re-enables asynchronous event triggers after a critical section of code.	5-80, 5-83
END	Ends a program and closes all files.	5-3, 5-4
EOF()	Tests for the end of a file.	5-68, 5-69
ERASE	Frees the memory used by an array.	5-3, 5-5
ERL()	Returns the line number where the error occurred, or the closest line number before the line where the error occurred.	5-95
ERR()	Returns the runtime error code for the most recent error.	5-95
ERROR	Simulates an occurrence of an error.	5-95
EVENT	Allocates a keyboard event or timer event.	5-80, 5-84
EVENTON	Returns the state of the event.	5-80, 5-83
EXP()	Returns e raised to a specified power, where e is the base of natural logarithms.	5-30, 5-32
FIELD	Defines the structure of records to be used in indexed-sequential and random-access file buffers.	5-65, 5-68
FLUSH	Discards received data in the BIOS serial input buffer.	5-53, 5-56
FOR .... NEXT	Repeats a section of the program the specified number of times.	5-20, 5-23
GET	Reads a record from the random-access or indexed-sequential file.	5-65, 5-69
GOSUB	Branches to a specified line number with intent to return to the next line.	5-20, 5-23
GOTO	Branches unconditionally to the specified line number.	5-20, 5-24
HEX$()	Returns a hexadecimal string representation of a number.	5-36, 5-37
IF .... THEN	Executes the sub-statement depending on specified conditions.	5-20, 5-24
INDEXED	Identifies a file as an indexed-sequential file and which field in the record is the index key.	5-65, 5-70

Command	Usage	Page
INKEY$	Returns a single keystroke from either the keyboard or keypad as a string.	5-45, 5-47
INPUT	Reads input from the keyboard, serial port, or a file.	5-45, 5-47, 5-53, 5-56, 5-65, 5-71, 5-80, 5-85
INSTR	Returns the position of the first occurrence of a string in another string.	5-36, 5-38
INT()	Returns the largest integer less than or equal to a numeric expression.	5-30, 5-33
IPD	Converts a double float representation of an IP address to a dotted string representation of an IP address	5-97, 5-101
IPS	Converts the dotted string representation of an IP address to a double for storage in Shared Data.	5-97, 5-101
JULDATE	Converts a date-time string: "mm-dd-yyyyHH:MM:SS" to a double precision Julian date.	5-92, 5-93
KEYSRC	Reports latest keystroke read by JagBASIC through INPUT or INKEY$ commands.	5-45, 5-50
KILL	Deletes the specified file from the JAGXTREME RAMDISK.	5-3, 5-5
LCASE$	Convert a string to lower case.	5-36
LEFT$()	Returns a specified number of leftmost characters in a string.	5-36, 5-38
LEN()	Returns the number of characters in a string or the number of bytes required to store a variable.	5-36, 5-39
LET	Assigns the value of an expression to a variable.	5-11, 5-17
LINE INPUT #	Reads sequentially all characters of an entire line (up to 80 characters) without delimiters from a sequential file up to the next carriage return into a string variable.	5-65, 5-71
LIST	Lists all or part of a program to the LPRINT device.	5-3, 5-5
LISTEN	Initializes TCP/IP to begin queuing the connection requests for the host port.	5-97, 5-101
LOAD	Loads a file (filename.bas) from the RAMDISK into memory.	5-3, 5-6
LOC()	Returns the current position within a file.	5-65, 5-71
LOF()	Returns the length of a file.	5-65, 5-71
LOG	Returns the natural logarithm of a numeric expression.	5-30, 5-33
LPRINT	Outputs data to a JAGXTREME LPRINT serial port device.	5-53, 5-57
LPRINT USING	Prints formatted output on the LPRINT device.	5-53, 5-58
LSET	Moves data into a random-access file buffer (in preparation for a PUT statement) and left-justifies the value of a string variable.	5-71, 5-74
LTRIM$	Removes spaces at the beginning of a string.	5-36, 5-39
MID$()	Returns part of a string.	5-36, 5-39
MSET$	Inserts one string into another string at a specified position. Overwrites the existing characters so the length of the string remains the same.	5-36, 5-40
MKI$, MKS$, MKD$	Convert numbers to numeric strings that can be stored in FIELD statement string variables.	5-65, 5-73

Command	Usage	Page
NEW	Clears the current program and all variables from memory.	5-6
NEWLADDER	Clears the ladder that is used by the ladder logic processor in the JAGXTREME operating system.	5-80, 5-86, 5-88
NEXTLINE	Displays next line number to execute or sets a "new" next line	5-3, 5-6
NEW	Clears the current program and all variables from memory.	5-3, 5-6
OCT$()	Returns an octal string representation of a number.	5-36, 5-40
ON ERROR GOSUB	Enables error handling and, when a run time error occurs, directs your program to an error handling routine.	5-95, 5-96
ON ERROR GOTO	Enables error handling and, when an error occurs, directs your program to an error handling routine.	5-95, 5-977
ON EVENT GOSUB	Enables you to asynchronously monitor an event. Defines the Event Service Routine.	5-81, 5-86, 5-89
OPEN	Accesses a file or prepares a serial port for use as a file device.	5-53, 5-59, 5-65, 5-73
OPTION BASE	Declares the minimum value (0 or 1) for array subscripts.	5-11, 5-17
OR	Used as a logical operator in a decision statement to establish two possible conditions, of which only one needs to be met.	5-20, 5-25
PADC$	Add pad characters to beginning and end of a string	5-36, 5-41
PADL$	Add pad characters to beginning of a string	5-36, 5-41
PADR$	Add pad characters to end of a string	5-36, 5-42
PRINT	Writes data to the lower display, to a sequential file, or outputs data to the specified serial port.	5-45, 5-50, 5-53, 5-61, 5-66, 5-74
PRINT #	Outputs data to a sequential file, or outputs data to the specified serial port.	5-53, 5-63, 5-66, 5-76
PRINT USING	Writes formatted output to the JAGXTREME display or to a file.	5-45, 5-50, 5-53, 5-61, 5-66, 5-74
PUT	Writes a record to a random-access file or an indexed-sequential file.	5-66, 5-76
RANDOMIZE	Initializes the random-number generator.	5-30, 5-33
READ	Reads values from a DATA statement and assigns them to variables.	5-11, 5-18
RECV$	Allows the JagBASIC to receive data over an established connection.	5-97, 5-101, 5-103
REM	Allows adding any comments or reference remarks to the code listing.	5-3, 5-7
RESETJAG	Re-initialize JAGXTREME by forcing power-up cycle.	5-20, 5-26
RESETKEYS	Sets the JagBASIC keyboard parameters back to a known state.	5-3, 5-6
RESTART	Clears the JagBASIC execution stacks and sends program control to the first line of the current program.	5-20, 5-26
RESTORE	Allows DATA statements to be reread from a specified line.	5-11, 5-18
RETURN	Used in conjunction with GOSUB, indicates that the subroutine is complete.	5-20, 5-26
RIGHT$()	Returns a specified number of rightmost characters in a string.	5-36, 5-42

Command	Usage	Page
RND	Returns a single-precision random number between 0 and 1.	5-30, 5-33
RSET	Moves data into a random-access file buffer (in preparation for a PUT statement) and right-justifies the value of a string variable.	5-66, 5-77
RTRIM$	Remove spaces from the end of a string.	5-36, 5-42
RUN	Executes the current file in memory.	5-3, 5-7
RUNGAND	Adds a ladder rung and takes two inputs AND's them together, and outputs the value.	5-80, 5-87
RUNGANDNT	Adds a ladder rung and takes two inputs AND's them together, and outputs the inverse value.	5-80, 5-88
RUNGMOV	Adds a new rung to the ladder and commands the ladder logic processor to continually move the value of one shared data variable into another.	5-81, 5-88
RUNGMVNOT	Adds a new rung to the ladder and commands the ladder logic processor to continually move the "not" or opposite value of one shared data variable into another.	5-81, 5-88
RUNGOR	Adds a ladder rung and takes two inputs OR's them together, and outputs the value.	5-81, 5-89
RUNGORNT	Adds a ladder rung and takes two inputs OR's them together, and outputs the inverse value.	5-81, 5-89
RZ	Initiates a ZMODEM file receive over serial port 1 into the RAMDISK file system.	5-3, 5-7
SAVE	Saves the current BASIC program in memory to the RAMDISK with the specified file name.	5-3, 5-7
SEND	Allows the JagBASIC to send data over an established connection.	5-97, 5-102
SGN	Returns a value indicating the sign of a numeric expression.	5-30, 5-34
SHOW	Displays the last line executed, the variable name and current value	5-3, 5-8
SIN()	Returns the sine of a specified angle expressed in radians.	5-30, 5-34
SLEEP	Suspends program execution for the of specified number of milliseconds.	5-92, 5-94
SOCKET	Creates a socket for a subsequent CONNECT command, which initiates a connection to a remote host using this socket.	5-97, 5-103
SOCKCLS	Allows the JagBASIC application to close an established TCP/IP connection.	5-97, 5-103
SOCKOPT	Makes a TCP/IP socket blocking or non-blocking.	5-97, 5-104
SORTREC	Identifies the file as an indexed sequential file and sorts records in the file.	5-6, 5-78
SPACE$()	Returns a string of spaces.	5-36, 5-43
SPC()	Skips a specified number of spaces in a PRINT or LPRINT statement.	5-53, 5-63
SQR()	Returns the square root of a numeric expression.	5-30, 5-34
STARTIME	Starts the timer, which specifies the length of the timer in milliseconds.	5-81, 5-90
STEP	Executes next line number after BREAK	5-3, 5-8
STOP	Terminates program execution and returns to command level.	5-3, 5-8

Command	Usage	Page
STOPTIME	Stops a running timer.	5-81, 5-90
STR$	Returns a string representation of a number.	5-36, 5-43
STRING$()	Returns a string of a specified length made up of a repeating character.	5-36, 5-43
SWAP	Exchanges the values of two variables that are variables of the same data type.	5-11, 5-19
SWITCHSUB	Performs a GOSUB call to the line specified in the variable.	5-20, 5-26
SWITCHTO	Performs a GOTO operation to the line specified in the variable.	5-20, 5-28
SZ	Initiates a ZMODEM file transfer over serial port 1 from the RAMDISK.	5-3, 5-9
TAB	Advances to the specified print position.	5-53, 5-64
TAN()	Returns the tangent of a specified angle expressed in radians.	5-30, 5-34
TIMEDAT$	Converts a double precision floating point Julian date number to a string: "mm-dd-yyyyJJ:MM:SS"	5-92, 5-94
TIMER	Returns a double precision floating point number that contains the elapsed time in seconds since 00:00:00 GMT, January 1, 1970.	5-92, 5-94
TIME$	Sets or returns the JAGXTREME system time.	5-92, 5-95
TRON, TROFF	Enables and disables tracing of program statements.	5-3, 5-9
UCASE$	Convert a string to upper case characters.	5-36, 5-44
VAL()	Converts a string representation of a number to a number.	5-36, 5-44
VARS	Prints a list of all variables to the LPRINT device.	5-3, 5-10
WAITEVENT	Suspends program execution until an event trigger causes program execution to resume.	5-81, 5-91
WATCH	Monitors variable during execution	5-3, 5-10
WHILE .... WEND	Repeats a section of the program until a specified logical condition is true.	5-20, 5-28
WIDTH	Assigns an output line width to the LPRINT device or a file.	5-53, 5-64
WIDTHIN	Allows you to dynamically reassign serial input length, as it is defined in OPEN.	5-53, 5-64
WRITE #	Writes data to the LPRINT device or to a sequential file.	5-66, 5-79
XOR	Used as a logical operator in a decision statement to establish two possible conditions, only one of which can be met.	5-20, 5-28

METTLER TOLEDO
1900 Polaris Parkway
Columbus, Ohio 43240
Phone: (US and Canada) (800) 786-0038
(614) 438-4511
(All Other Countries) (614) 438-4888

www.mt.com

16384600A
(10/01)