DuraTrax® /LaserLite®
LaserLite Pro/LaserLite Mx

Developer's Reference

GCO# 930 MN-DTL-06

Table of Contents

Introduction

This manual provides additional information for the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx software developer. This manual consists of eight chapters and three appendixes.

Chapter 1 contains information on the communications programs: **Vxcom**, **Download**, and **Videx Download**. These programs provide communication between the data collector and the computer.

Chapter 2 contains information on the monitor program that is the boot program that resides in the ROM of the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.

Chapter 3 contains information on the operating system files for the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx. The operating system must be installed on the data collector before it can collect data.

Chapter 4 contains information on the Application Builder Source Template (which generates source code in BASIC), including information on plug-ins and exporting the source file.

Chapter 5 contains information on the LaserLite Mx memory card and how to manage files on the memory card.

Chapter 6 contains information on DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx data files. These are the files of collected data that are transferred to the computer.

Chapter 7 contains information on the cross-reference file convert programs: **Vxcrfw.exe** and **Vxcrf.exe**. These programs convert a text file into a cross-reference file that can be used by a DuraTrax, LaserLite, LaserLite Pro, or a LaserLite Mx without a memory card.

Chapter 8 discusses the theory of operation for the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx data collectors.

Appendix A contains information on connecting the Base Station to a modem, Appendix B contains information on formatting LaserLite Mx memory cards, and Appendix C contains lists of error codes.

Chapter 1

Communications Programs

This chapter contains information on:

- The communications programs for the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.
- **Vxcom** for Windows communications program for DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.
- **Download** for DOS communications program for DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.
- **Videx Download** for Macintosh communications program for DuraTrax, LaserLite, and LaserLite Pro.
- Transferring files between the data collector and the computer.

Communications Programs Overview

The communications programs, **Vxcom** (Windows), **Download** (DOS), and **Videx Download** (Macintosh), provide communication between the data collectors and a computer. These programs allow you to transfer files between the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx data collectors and the computer. (Note: The LaserLite Mx is not Macintosh compatible.)

The communications programs interact with several programs or files used by the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx system; including the monitor program, operating system, commands file, application and cross-reference files, and data files. The following two diagrams show how these components interact for the DuraTrax, LaserLite, LaserLite Pro and the LaserLite Mx.

The following diagram illustrates how these components interact for the DuraTrax, LaserLite, and LaserLite Pro.

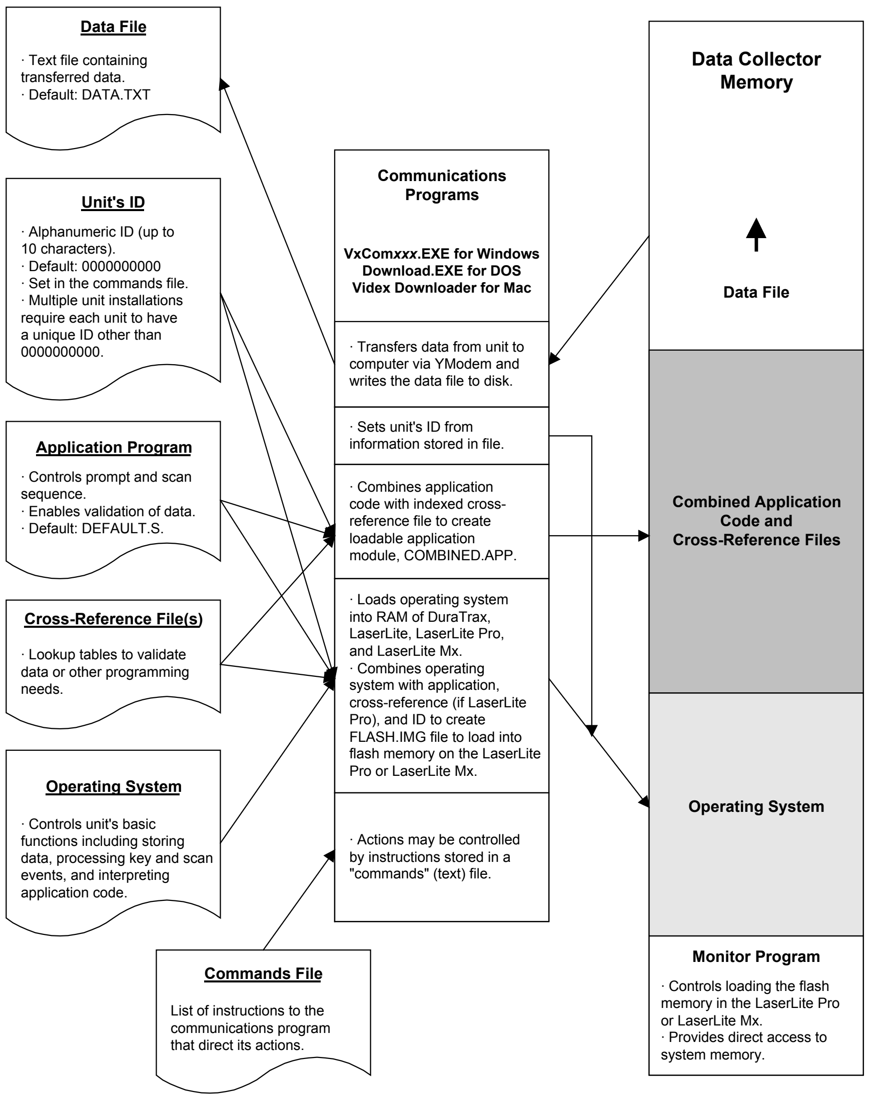

Data File
· Text file containing transferred data.
· Default: DATA.TXT
Unit's ID
· Alphanumeric ID (up to 10 characters).
· Default: 0000000000
· Set in the commands file.
· Multiple unit installations require each unit to have a unique ID other than 0000000000.
Application Program
· Controls prompt and scan sequence.
· Enables validation of data.
· Default: DEFAULT.S.
Cross-Reference File(s)
· Lookup tables to validate data or other programming needs.
Operating System
· Controls unit's basic functions including storing data, processing key and scan events, and interpreting application code.
Commands File
List of instructions to the communications program that direct its actions.
Communications Programs
VxComxxx.EXE for Windows
Download.EXE for DOS
Videx Downloader for Mac
· Transfers data from unit to computer via YModem and writes the data file to disk.
· Sets unit's ID from information stored in file.
· Combines application code with indexed cross-reference file to create loadable application module, COMBINED.APP.
· Loads operating system into RAM of DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.
· Combines operating system with application, cross-reference (if LaserLite Pro), and ID to create FLASH.IMG file to load into flash memory on the LaserLite Pro or LaserLite Mx.
· Actions may be controlled by instructions stored in a "commands" (text) file.
Data Collector Memory
Data File
Combined Application Code and Cross-Reference Files
Operating System
Monitor Program
· Controls loading the flash memory in the LaserLite Pro or LaserLite Mx.
· Provides direct access to system memory.

Figure 1-1 Communications Programs and DuraTrax, LaserLite, and LaserLite Pro Interactions with Programs and Files

The following diagram illustrates how these components interact for the LaserLite Mx.

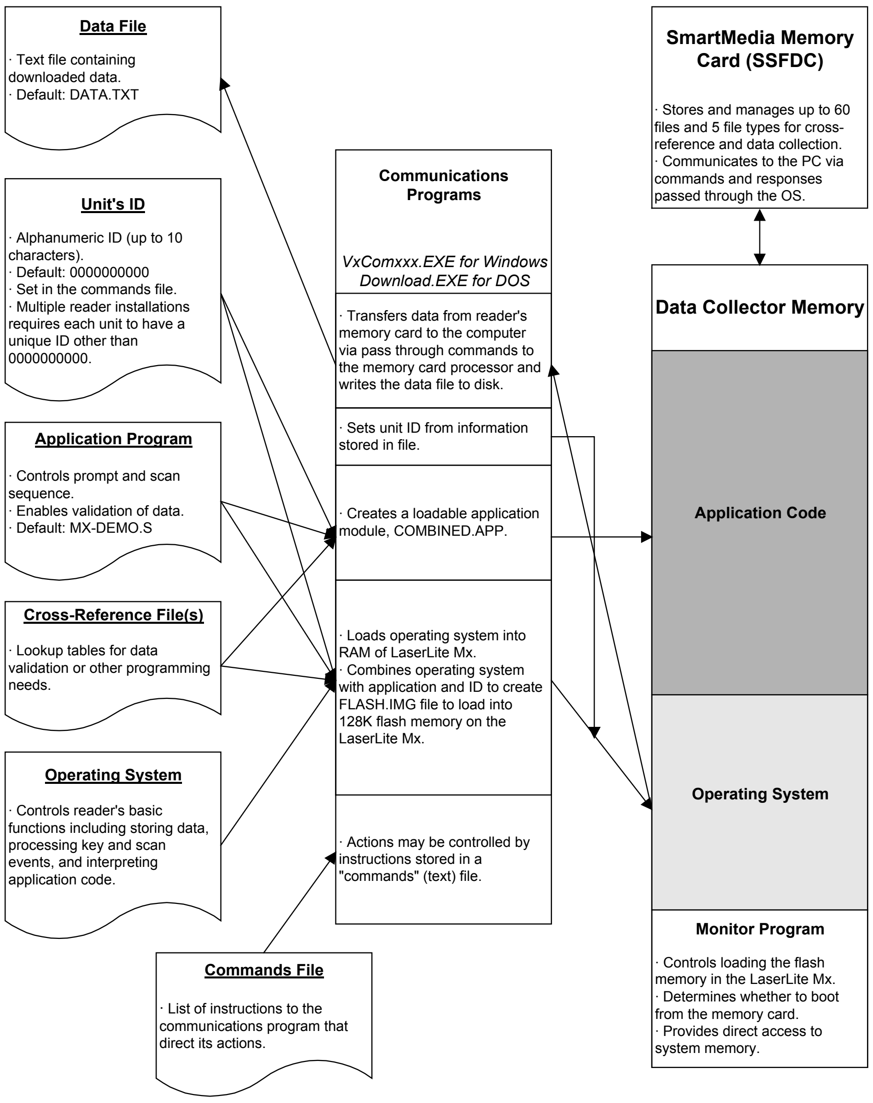

Data File
· Text file containing downloaded data.
· Default: DATA.TXT
Unit's ID
· Alphanumeric ID (up to 10 characters).
· Default: 0000000000
· Set in the commands file.
· Multiple reader installations requires each unit to have a unique ID other than 0000000000.
Application Program
· Controls prompt and scan sequence.
· Enables validation of data.
· Default: MX-DEMO.S
Cross-Reference File(s)
· Lookup tables for data validation or other programming needs.
Operating System
· Controls reader's basic functions including storing data, processing key and scan events, and interpreting application code.
Commands File
· List of instructions to the communications program that direct its actions.
Communications Programs
VxComxxx.EXE for Windows
Download.EXE for DOS
· Transfers data from reader's memory card to the computer via pass through commands to the memory card processor and writes the data file to disk.
· Sets unit ID from information stored in file.
· Creates a loadable application module, COMBINED.APP.
· Loads operating system into RAM of LaserLite Mx.
· Combines operating system with application and ID to create FLASH.IMG file to load into 128K flash memory on the LaserLite Mx.
· Actions may be controlled by instructions stored in a "commands" (text) file.
SmartMedia Memory Card (SSFDC)
· Stores and manages up to 60 files and 5 file types for cross-reference and data collection.
· Communicates to the PC via commands and responses passed through the OS.
Data Collector Memory
Application Code
Operating System
Monitor Program
· Controls loading the flash memory in the LaserLite Mx.
· Determines whether to boot from the memory card.
· Provides direct access to system memory.

Figure 1-2 Communications Programs and LaserLite Mx Interactions with Programs and Files

The following paragraphs summarize the purpose of each program and file.

Monitor Program: The **monitor program** is the boot program that resides in processor memory. It is the lowest level of software in the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx system that presents an RS-232 interface. It provides direct access to system memory and controls loading an initial system executable program. The monitor program is described in Chapter 2.

Operating System: The **operating system** (**OS**) controls the data collector's basic functions including storing data, processing keypresses and scan events, and interpreting application code. The operating system software filename always ends with a **.OS** extension. The operating system software is described in Chapter 3.

Commands File: A **commands file** is a list of instructions to the communications program. You may create a commands file from a text editor or generate it from a custom application program. Commands file details are discussed on pages 25–42.

ID: By default, the **ID** of each data collector is **0000000000**. You may assign a unique name or number (up to 10 characters) to a data collector. The ID is stored by the operating system. A unique ID enables the communications program to isolate and control a single data collector from others connected to the same serial port on a computer, thus enabling communications with multiple data collectors. The ID is created and sent to the computer via the commands file.

Application Program: An **application program** controls the data collection process, including prompting, validation of data, and the data file format. An application program is created with Application Builder, compiled from the generated BASIC source code, then stored on the computer as a ***.S** file. Applications compiled for DuraTrax, LaserLite, and LaserLite Pros, but loaded into a LaserLite Mx along with a LaserLite Pro operating system (***.OS**), will cause the LaserLite Mx to behave exactly like a LaserLite Pro. *(Note: The asterisk * represents the application filename.)* See the *Application Builder Manual* for more information on Application Builder.

Cross-reference File: A **cross-reference** file is a lookup table used by an application for data validation, configuration, or other needs. An application may refer to one or more cross-reference files. This manual will refer to two types of cross-reference files: ***.CRF cross-reference files** and **text cross-reference files**.

***.CRF cross reference files (DuraTrax, LaserLite & LaserLite Pro)**
Cross-reference files for the DuraTrax, LaserLite, LaserLite Pro must have a **.CRF** extension. They may be created using Application Builder or by converting an existing cross-reference text file using either the **Vxcrf.exe** or **Vxcrfw.exe** utility programs (refer to Chapter 7). Up to 31 cross-reference files of this type can be called from an application. **Vxcom** or **Download** communications programs automatically merge ***.CRF** cross-reference files with the application code prior to loading into a DuraTrax, LaserLite, or LaserLite Pro so that the application can access them during runtime.

Text cross-reference files (LaserLite Mx)
Cross-reference files intended for the LaserLite Mx memory card must be text files with tab-delimited fields and no quotes. **Vxcom** or **Download**, based on instructions in a commands file, send these files to the LaserLite Mx one record at a time. The LaserLite Mx then indexes each record it receives to a hash table to enable fast lookups on the key field. The LaserLite Mx operating system also supports *.**CRF** files but they must be merged and transferred with the application (***.S**) file the same as DuraTrax, LaserLite, and LaserLite Pro. Since the LaserLite Mx data management system provides more comprehensive file handling capabilities, this manual presumes that all cross-reference files for the LaserLite Mx will be directed to its memory card.

Image File: *(LaserLite Pro and LaserLite Mx only)* An image file is an exact copy of the merged operating system, application, unit ID, and any ***.CRF** cross-reference files that reside in the RAM of the LaserLite Pro and LaserLite Mx. An image file can be created by **Vxcom** or **Download** and transferred to the 128K flash memory of the LaserLite Pro or LaserLite Mx. Once the image file is in the flash memory, it serves as a backup copy of the operating system and application should the copy in RAM be lost or damaged. In that event, the contents of the flash memory may then be copied to RAM. The image file can also serve as a system boot file when it is loaded onto the LaserLite Mx memory card using the **MXFORMAT** utility.

Vxcom and **Download** use the **-f** and **-k** arguments to create and control the image file. The **-k1** and **-k2** arguments enable you to build the image file from the applicable files listed on the command line and save it to disk. See the **-f** and **-k** arguments on pages 15–16 for more information.

Data File: A **data file** is the collection of all data collected by the data collector since the last data transfer. The communications program extracts the collected data from the data collector and writes it to the computer in the form of an ASCII text file. Data file structure and options are described in Chapter 6 of this manual. The name of the data file is defined by the application. Videx uses **DATA.TXT** as the default data filename. The data filename can be changed by modifying the application's BASIC source code.

Communications Programs for Windows, DOS, and Macintosh

The communications program for DOS is **Download** and the communications program for Windows 95/98/NT is **Vxcom**; these two programs can be used with a DuraTrax, LaserLite, LaserLite Pro, or a LaserLite Mx.

The communications program for Macintosh is **Videx Download**; it can be used with a DuraTrax, LaserLite, LaserLite Pro, or a LaserLite Mx with a LaserLite Pro application and operating system. The LaserLite Mx operating systems and applications are not Macintosh compatible.

The following sections describe the three communications programs.

Download for DOS

The communications program for DOS computers is **Download.exe**; it can be used with a DuraTrax, LaserLite, and LaserLite Pro.
LaserLite Mx requires **Download.exe** version 2.0.0 or later.
Download.exe executes a series of commands from a commands file.

The syntax for the **Download.exe** command line is:

DOWNLOAD [app.s] [REF.CRF] [sys.os] [cmd.txt] {arguments}

or

DOWNLOAD [image.img] [cmd.txt] {arguments}

The parameters are listed on one program line and may be listed in any order. The parameters and defaults are identical to the ones used in the **Vxcom** program; they are described on pages 13–19.

Vxcom for Windows

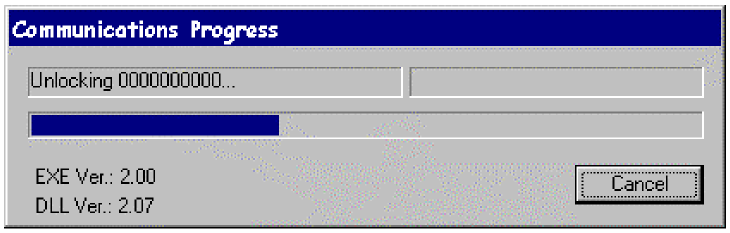
The communications program for Windows 95/98/NT is **Vxcom**; it can be used with a DuraTrax, LaserLite, and LaserLite Pro. LaserLite Mx requires **Vxcom** version 2.00 or later.

The original **Vxcom** filename was **Vxcomm.exe**; newer versions of **Vxcom** have names in the form **Vxcom*xxx*.exe**, where ***xxx*** corresponds to the version number (for example, **Vxcom200.exe**). These files are accompanied by a DLL with a name in the form **Vxcom*xxx*.dll**. Most recently, **Vxcomm.dll** version 2.1 was created that provides the enhancements in **Vxcom200** and also supports the Application Builder program.

If you have previously used the Application Builder software, you may have used **Vxcom** to transfer the data file to the computer. **Vxcom** also transfers files from the computer to the data collector.

The **Run** command is used from the Windows **Start** menu to pass parameters to **Vxcom**. The parameters and defaults are listed on pages 13–19. Alternatively, you may start the **Vxcom** program by double-clicking its icon or by dragging and dropping a combination of the commands file, application file, CRF file, and operating system file icons onto the **Vxcom** icon.

Vxcom displays a progress bar window (Figure 1-3) to report the status of the file transfer.


Communications Progress

Unlocking 0000000000...

EXE Ver.: 2.00

DLL Ver.: 2.07

Figure 1-3 Vxcom Progress Window

Vxcom executes a series of commands from a commands file. See the **Commands File** section beginning on page 25 for more information on the commands file.

The syntax for the **Vxcom** command line is:

> **Vxcom*xxx*** [app.s] [REF.CRF] [sys.os] [cmd.txt] {arguments}
>
> *or*
>
> **Vxcom*xxx*** [image.img] [cmd.txt] {arguments}

The parameters are listed on one line and they may be listed in any order. If the files listed on the command line are not in the same folder as **Vxcom**, you must also include the file path.

Download and Vxcom Parameters

Following are descriptions of the parameters for **Download** and **Vxcom**:

app.s	The compiled application program: either an application created with Application Builder, with Videx BASIC, or the default application **Default.s**. Triggered by the **.S** extension. This parameter is optional. (Note: If an ***.OS** file is passed as a parameter, but a ***.S** file is not listed, nor can the **Default.s** file be located, then the file transfer process terminates.) To avoid the error, the **Default.s** file must reside in the same folder as the **Vxcom** or **Download** communications programs.
REF.CRF	This is the application program's ***.CRF** cross-reference files *(if the application uses* ***.CRF** *files)*. Triggered by the **.CRF** extension. If the application uses more than one cross-reference file, the ***.CRF** filenames must be separated by a space. This parameter is optional, but if the application program uses ***.CRF** cross-reference files, they must be listed for the application to perform properly.

***LaserLite Mx Memory Card Note:** Text cross-reference files for the LaserLite Mx memory card must be transferred to the memory card with a commands file. See pages 25–42 for information on the commands file, see Chapter 5 for information on using LaserLite Mx memory cards, and see pages 137–138 and 147 for examples of transferring a cross-reference file to a memory card.

sys.os	The operating system software for the data collector. Triggered by the **.OS** extension. This parameter is optional unless you have reset the data collector to monitor mode. (Note: The exact name of the operating system software will change as new versions are released, but will always end with a **.OS** extension. Each product's operating system will always begin with the same letters: **Trax** for DuraTrax, **Lite** for the LaserLite; **Pro** for the LaserLite Pro; and **Lmx** for the LaserLite Mx.)
image.img	The image file is a combination of the operating system, application program, and ***.CRF** cross-reference files that can be loaded into the flash memory of the LaserLite Mx or LaserLite Pro. Triggered by the **.IMG** extension. This parameter can be used instead of, but not with the ***.S**, ***.CRF**, and ***.OS** parameters. The image file is controlled by the **-f** and **-k** arguments described on pages 15–16. (Only available for versions 1.38 or later of **Vxcom** and **Download**.)
cmd.txt	The commands file is a text file containing commands to be performed by the communications program. Triggered by **.TXT**, **.VDX**, or no extension. (See the commands file description and defaults on pages 25–42.) This parameter is optional unless you are transferring files to or from the LaserLite Mx memory card. (Note: This file can also change a data collector's ID.)

{arguments} include:

-d*n* Indicates IR device to use (0–1); ***n*** is the device number. Use 0 for a Videx Base Station or if your computer has a built-in IR transceiver, and 1 for a JetEye. Default is 0. (Note: This argument's configuration numbers changed with version 1.38. Prior versions used **-d2** for a built-in IrDA transceiver.)

0 - (default) Prepares Base Station or computer's built-in IrDA transceiver for communications with data collector. (Note: DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx support the IrDA physical layer only. Computers with built-in IrDA transceivers must disable the link access protocol and the link management layers before using **Vxcom** or **Download**.)

1 - For use with JetEye PC IR station; pulls DTR low.

-f*n* This argument is for LaserLite Pro and LaserLite Mx only, and works only in conjunction with the **-k** argument. It indicates if the image file (***.IMG**) should be loaded into the flash memory of the LaserLite Mx or LaserLite Pro; ***n*** is the option number (0–1). Default is 0. (This argument is only available for versions 1.38 or later of **Vxcom** and **Download**.)

0 - (default) Do not load to flash memory.

1 - Load image file to flash memory.

-k*n* This argument works in conjunction with the **-f** argument. Indicates if the image file (***.IMG** file) is created, sent to the data collector, and erased from the computer; *or* if the image file is created, sent to the data collector, and a copy of the ***.IMG** file is kept on the computer; *or* if the image file is created and kept on the computer; ***n*** is the option number. Default is 0. (This argument is only available for versions 1.38 or later of **Vxcom** and **Download**.)

0 - (default) Sends file (***.IMG**) to data collector then erases file from computer.

1 - Sends file to data collector and saves ***.IMG** file on computer.

2 - Keeps file on computer; does not send file to data collector.

-p*n* Indicates serial port to use (1–4); ***n*** is the port number. You can use ports 1, 2, 3, or 4. Default is 1.

-q*n* This argument is only for **Download.exe** versions 1.38 and later. Indicates if the communications program displays its messages or operates in the background (0–1); ***n*** is the option number. Default is 0.

0 - (default) Messages displayed on computer screen.

1 - Quiet; no messages. Operates in background.

-s*n* This argument indicates if the communications result code and unit ID are kept in the **Status.txt** file or not (0–2); ***n*** is the option number. Default is 0. (This argument is only available for versions 1.38 or later of **Vxcom** and **Download**.)

0 - (default) Does not record status.

1 - Writes status to file; overwrites current file.

2 - Writes status to file; appends to current file.

Note: No status message is written for an **Unlock** command timeout error. Pressing the <Escape> key will terminate the communications program.

The format of the **Status.txt** is as follows:

Line 1 – Name and version of calling program with the date and time executed.
Line 2, 3, ... (up to the number of units contacted in session) – Unit's ID (if contacted), error number and description, or communication result, date, and time.

Following is an example of a **Status.txt** file, followed by a table that describes the status codes.

Status.txt File Example

```
Download v. 1.38  09/16/97  13:33:09
DURATRAX, 16001, I gave up while waiting for a character.
VXComm v. 1.38  09/16/97  13:47:42
0000000000 No data.  13:47:54
VXComm v. 1.38  09/16/97  13:48:00
DURATRAX  Transferred to DATA.TXT  13:48:12
VXComm v. 1.38  09/19/97  11:58:41
, 18007, There was an error talking to a serial port.
Download v. 1.38  09/23/97  10:08:26
0000000000  Transferred to DATA.TXT  10:09:08
DURATRAX  Transferred to DATA.TXT  10:09:23
```

Status.txt Codes

Code	Description
16001	Gave up waiting for a character.
16002	Aborted transmission.
16003	Got out of sync with other party in transmission.
16004	Keep missing parts of packets; there may be too much noise on line.
16005	Received the start of the packet, but rest of packet is badly formed.
16006	Received a well formed packet, but the check digits were wrong. May have noise on the line.
18000	Cannot open the serial port.
18001	Unknown command in the commands file.
18002	Error in communications.
18003	**C** and **I** commands in commands text file require identifiers.
18004	No such serial port.
18005	Error accessing a file.
18006	Not enough memory to continue.
18007	Error talking to the serial port.

Code	Description
18008	One of the command line arguments in the commands file does not make sense.
18009	Loops in the commands file are nested too deeply.
18010	The file size exceeds the device’s capacity.
18011	There is no application file to load with the **OS**.
18012	You must specify either an **OS** file or an image file to load into flash.
18013	Image files (***.IMG**) can only be loaded into flash.
18014	The structure of a cross-reference file is corrupted.
21000	Operation successful.
21001	Unrecognized memory card. This version recognizes Toshiba’s SSFDC 2, 4, and 8 MB memory cards.
21002	Unrecognized memory card.
21003	Syntax error.
21004	CRC of command did not match.
21005	Unknown command.
21006	Missing parameters for this command.
21007	Incorrect parameters for this command.
21008	Binary file.
21009	Incorrect file type.
21010	Too much data in the command.
21022	No ID file.
21023	No data.
21031	No such file exists.
21032	No file opened.
21033	Too many files (>60 files).
21034	The page has already been written to four times. (Note: Toshiba only allows you to write to a page four times.)
21035	No such record exists.
21036	End of file.
21037	Beginning of file.
21038	Timeout occurred.
21039	Memory full.

Code	Description
21040	Write/protect encountered.
21041	Record too large (>1024 bytes).
21042	Field too large (>255 bytes).
21043	Some of the physical functions are locked to avoid data corruption. Try: **N &0129** to unlock.
21051	File management error (control data was changed).
21052	Sequential file corrupted (by accidental power failure).
21053	Data corrupted.
21054	CRC of boot file did not match (program data may be corrupted).
21061	Memory card program failed (card may be worn out).
21062	Block erase failed (card may be worn out).
21063	Unknown memory card format.
21064, 21065	First block of memory is bad (broken card).
21066	Too many bad blocks (card may be damaged).
21067	Most of the reserved control blocks are bad (card worn out).
21068	The memory card has been erased more than 32,768 times (each **K &1092** or **N &1092** counts as one erasing).

Windows DLL for Vxcom

Vxcomm.dll (version 2.10)

Vxcomm.exe calls a 32-bit dynamically linked library (DLL) with functions that invoke the communications routines. This DLL is used by Application Builder to transfer an application to the data collector, and can also be used by developers who are integrating Videx data collectors into their Windows application. The header file **vxcomm.h** documents the C calling conventions. **Vxcomm.dll** has a standard export function **Vxcomm_vb**. (Note: If you have an earlier version of **Vxcomm.dll**, visit the Videx website at **www.videx.com** to obtain the latest version of **Vxcomm.dll**.)

This is an example of the function declaration from Visual Basic 6.0:

```
Declare Function vxcomm_vb Lib "vxcomm.dll" _
  (ByVal owner As Integer, _
  ByVal cmd As String, _
  ByVal sys As String, _
  ByVal app As String, _
  ByVal crfs As String, _
  ByVal port As Integer, _
  ByVal config As Integer) _
  As Integer
```

This is an example of a call from a form:

```
Private Sub Command0_Click()
        Dim x
        x = vxcomm_vb(pNull, command_var, sys_var, app_var, _
                crf_var, port_var, config_var)
        MsgBox (x)
End Sub
```

The parameters are defined below:

```
Vxcomm_vb (
        HWND owner              /*parent of progress window, if declared
                                as Null, progress window is parent of
                                top level window*/
        char const * cmd        /*name of commands file or NULL*/
        char const * sys        /*operating system filename or NULL*
        char const * app        /*application filename or NULL*/
        char const * crfs       /*comma delimited list of crf files*/
        long int port           /*port number (1-4)*/
        long int config         /*configuration, see below*/
);
```

The **config** parameter uses a bit-mapped scheme to pass a set of Boolean parameters (communications options). Previous versions of **Vxcomm.dll** used the **config** parameter to specify the serial mode to use, i.e., Videx downloader or JetEye IR transmitter. This functionality is preserved for backward compatibility. The communication options that are supported by version 2.10 of **Vxcomm.dll** via the **config** parameter are defined in the following table:

serial_type	Add 0 to **config** for Videx downloader; add 1 to **config** for JetEye IR transmitter; add 2 to **config** for standard serial.
flash	Add 0 to **config** to load to RAM; add 16 to **config** to load to flash.
save_temp	Add 0 to **config** for do not save temporary files; add 32 to **config** to save temporary files.
dont_load	Add 0 to **config** to load files into data collector; add 64 to **config** to not load files into data collector (effective only if saving temporary files).
quiet	Add 0 to **config** for not quiet; add 128 to **config** for quiet mode. This option is only present for code compatibility and has no effect on the execution of the function.
status	Add 0 to **config** for no status file; add 256 to **config** to create a status file.
stat_append	Add 0 to **config** to overwrite status file; add 512 to **config** to append to status file (effective only if creating a status file).
half_duplex	Add 0 to **config** for full duplex mode; add 1024 to **config** for half duplex mode.

If the **save_temp** option is enabled, the application or image file that is created by **Vxcomm.dll** is preserved rather than erased. If an application and cross-reference files are passed to **Vxcomm.dll**, the name of the file that is created is **combined.app**. If an operating system, application and cross-reference files are passed to **Vxcomm.dll**, the name of the created file is **flash.img**. If loading to the flash memory on the LaserLite Pro or the LaserLite Mx instead of to RAM, it is recommended that the program determine whether the image file (**flash.img**) will fit into the flash memory. The **check_file_size** function is provided to do this. The declaration from Visual Basic 6.0 is:

```
Declare Function check_file_size Lib "vxcomm.dll" _
        (ByVal filename as String, _
        ByVal max_size as Long) _
        as Integer
```

where **filename** is the name of the file to be loaded to flash memory (generally **flash.img**), and **max_size** is the maximum size (in bytes) of a file which is allowed to be loaded to flash (generally 128K or 131072). This function returns 0 if the size of **filename** is less than **max_size**; otherwise it returns 18010 (file size exceeds the device’s capacity).

Pseudo-code for creating an image file, checking its size, and then loading it into the data collector follows:

```
x = vxcomm_vb(pNull, "", sys, app, crfs, port, 112)  'create image file

If (x) Then
   MsgBox ("Error creating image")
Else
   y = check_file_size("flash.img", 131072)      'check if size>128K
   If (y) Then
      MsgBox ("Image too big")
   Else
      z = vxcomm_vb(pNull, cmd, sys, app, crfs, port, config)
                                                 'load software
      If (z) Then
         MsgBox ("Communications error")
      End If
   End If
End If
```

Videx Download for Macintosh

The communications program for Macintosh computers is **Videx Download**. **Videx Download** can be used with a DuraTrax, LaserLite, or LaserLite Pro. (Note: The LaserLite Mx operating system and applications are not Macintosh compatible.)

Videx Download executes a series of commands from a commands file. There are four types of documents that can be sent to **Videx Download**; they are: commands file (text file), operating system software (**OS**), application files, and cross-reference files.

To use **Videx Download**, drag and drop one or more files onto it. If you are using it as a server, send it an **Open Document** event, with the list of the files you want transferred. Only one commands file is recognized per **Open Document** event. If **Videx Download** fails to execute any of the commands, it quits immediately and returns an error code in the **Open Document** event.

Videx Download Interface (Macintosh only)

If **Videx Download** is run without any documents (it's sent an **Open Application** event), then it runs like a normal application. It has a **Settings** menu, which lists the available serial ports.

Following are activities that can be performed by **Videx Download**:

To change the serial port...	run **Videx Download**, choose **Settings**, and select a different serial port.
To transfer the data file...	double-click the **Download Data** icon.
To install new operating system software...	drag the operating system file and an application file to the **Videx Download** icon. (If unique IDs are required for each data collector, a commands file that changes the ID can also be dragged to the icon at the same time.)

Important Note: You must drag an application (either **Default.s** or an Application Builder application) along with the operating system to the **Videx Download** icon.

Communications Program Commands File

A commands file is an ASCII text file that lists the actions that the communications program should perform. The communications program attempts to execute the commands sequentially from the top to the bottom of the file. If it encounters an error, the program automatically stops executing and notifies the user of the error.

To be recognized by the communications program, the commands file must have a **.TXT** extension, a **.VDX** extension, or no extension. You can create a commands file with a text editor or generate it from a custom application program. If the communications program encounters more than one commands file, it only opens and attempts to execute from the last one.

A commands file typically includes commands to send a compiled application (***.S** file), associated ***.CRF** or text cross-reference files, receive the data file, set the clock to the computer's time, resend the operating system software (***.OS** file), change the ID, and so on. In addition, commands files support a form of looping that allows transferring data from multiple data collectors. The looping commands are discussed later in this chapter.

The communications programs have a built-in, default list of commands that they reference in the absence of an external commands file. Following is a description of the commands contained in the default commands file:

Default Commands File	**Description of Command Line**
`I 0000000000`	Unlocks the data collector. This is the **Unlock** command that signals an exit event to the application and puts the data collector into communications mode.
`M1 Downloading Data`	Puts message on line 1 of the data collector's display.
`S`	Instructs data collector to send data file.
`Z`	Instructs data collector to clear data file.
`M1 Loading Software`	Puts another message on line 1 of the display.
`R`	Instructs the data collector to prepare to receive a file or files.
`T`	Sets the clock in the data collector to match the clock in the computer.
`L`	Locks the data collector and puts it into sleep mode for minimum power consumption.

Commands

Any of the following commands can be listed in the commands file in any order; however, it is important to create the list of commands in an order that matches an expected flow of operation for the data collector. For example, the data collector must be unlocked before it can accept any of the other commands in the list, so each commands file list typically begins with an **I** (**Unlock**) command. Also, important data should be transferred from the unit prior to loading new operating system, application, or cross-reference files.

If any command generates an error, the communications program stops executing and returns the error. The exception to this is errors encountered in the looping commands described on pages 28–42.

Vxcom or **Download** versions 2.0 or later are required to support commands for the LaserLite Mx. Refer to pages 130–140 to work with the LaserLite Mx.

Note: A space is required between the command and the argument.

' An apostrophe (') indicates a comment. The communications program ignores any line in the commands file that begins with an apostrophe.

C *id* Set or change the data collector's ID to the given ID. If an ID is not given, the command fails.

D *<file type> <filename>* *(LaserLite Mx only)*
Delete a file from LaserLite Mx memory card.

F *[<output filename>]* *(LaserLite Mx only)*
List the LaserLite Mx memory card file management report.

G Run the current application using the operating system **GO** command.

I *id* Send an **Unlock** command to the data collector using the given ID. This command is sent every 5 seconds for 35 seconds.

K *(LaserLite Mx only)*
Remove any deleted files from the LaserLite Mx memory card.

L Send a **Lock** command and put the data collector to sleep. When awoken with a keypress, immediately run current application.

M1 ***message***
Set display line 1 to the given message.

M2 ***message***
Set display line 2 to the given message.

R ***[<number of bins>] <file type> <filename>***
Prepare the data collector to receive a file. Using the **R** command without the *[<number of bins>] <file type>* parameters, tells the data collector to receive the listed operating system software, application, or flash image file, depending on the files that are included in the communications program. If an operating system, application, or flash image is not being transferred, this command doesn't do anything (but doesn't fail).

Using the **R** command with the *[<number of bins>]<file type> <filename>* parameters sends the listed file to the LaserLite Mx memory card. The *[<number of bins>]* parameter must be passed if the file does not already exist on the memory card. The *<file type>* and *<filename>* parameters are case sensitive.

S ***[<folder path>] <file type> <filename>***
Sends a file to the computer. Using the **S** command without the *<file type>* parameters tells the data collector to send its data file. The file is put in the given *[<folder path>]*. If no *[<folder path>]* is given, **Vxcom** and **Download** write the file to the current directory. **Videx Download** for Macintosh writes the file to the desktop. The file is named by the BASIC application in the **OPEN** statement.

For example:
```
OPEN "data.txt" FOR APPEND AS #0
```

If a file with that filename already exists in the communications folder, then the new data is appended to that file. If an error occurs during transmission of the data, then the destination file is truncated to its original size. Using the *<file type>* parameter sends the listed file from the LaserLite Mx memory card to the computer. The *<file type>* and *<filename>* parameters are case sensitive.

T	Set the time on the data collector to the computer's time (synchronizes the clocks).
Z	Clear (zap) the data in the data collector.

Note: The commands in this list approximate the operating system commands described in Chapter 3.

Looping Commands

You can also use the following looping commands in the commands file.
FOREACH...END, FORALL...END, FOREVER...END

Looping commands are provided for transferring data from multiple units, managing retries, and setting up a continuous loop operation. The purpose of a "loop" in a commands file is to enable a command or set of commands to be repeated until a certain criterion is met. The criterion is determined by the **FOREACH**, **FORALL**, or **FOREVER** statement. The communications program executes each command within the loop, beginning with the **FOREACH**, **FORALL**, or **FOREVER** statement and ending with the **END** statement. Loops may be nested within loops up to four levels deep.

Example:

```
FOREVER
    commands
         FOREACH
             commands
         END
END
```

If an operation fails within a loop, execution continues immediately at the end of the loop and a status message is written to the **STATUS.TXT** file if the **-s1** or **-s2** parameter was used in the communications program's command line.

Note: No status message is written for an **Unlock** command timeout error. Pressing the <Escape> key will terminate the communications program.

When using looping commands to transfer the data from data collectors in a multiple Base Station set up, you must give each data collector a unique ID; this allows the computer to access each data collector and verify that it has transferred the data from each one. Refer to your data collector's hardware manual for instructions on connecting multiple Base Stations to one computer.

Assigning IDs

Changing a Data Collector's ID with a Windows Computer

In Windows you can change a data collector's ID by first creating a commands text file, then clicking and dragging the file onto the **Vxcom** icon. **Vxcom** uses the commands file to unlock the data collector and change its ID.

During normal operation, a unit is given an ID of ten zeros. To change the ID, create a text file consisting of the following three command lines:

1) an **I** command, followed by a space, followed by ten zeros to unlock the unit;
2) a **C** command, followed by a space, followed by the new ID for the unit;
3) and an **L** command to relock the unit.

I 0000000000	Sends an **Unlock** command to the data collector. (Note: Ten zeros will unlock any data collector.)
C *new ID#*	Changes the unit's ID to the new ID. The ID can be any alphanumeric combination of up to ten characters in length.
L	Sends a **Lock** command and puts the data collector to sleep. When the unit is awakened with a keypress, it immediately runs the current application.

For example, to change a unit's ID to **12345**:

1. Use a word processing program to create a text file containing the following three lines:

   ```
   I 0000000000
   C 12345
   L
   ```

2. Save the file as a text file with a **.txt**, **.vdx**, or no extension and quit the program.

3. Place a data collector into the Base Station slot.

IMPORTANT: When sending a file to a data collector, it must be the ONLY data collector attached to the computer.

4. Click and drag the text file you created onto the **Vxcom** icon.

5. **Vxcom** executes the commands listed in the text file: it unlocks the unit, changes the ID to what is listed after the **C** command (in this case, 12345), and then relocks the unit.

6. Remove the data collector from the Base Station slot; it is ready to use.

To continue changing other data collector's IDs:

1. Reopen the text file.

2. Change the ID listed after the **C** command by selecting **12345** and typing in a new ID. (Note: There must be a space character between the **C** and the ID.)

3. Save and close the file.

4. Insert a data collector into the Base Station slot.

5. Click and drag the edited text file to the **Vxcom** icon.

6. **Vxcom** executes the commands listed in the text file: it unlocks the unit, changes its ID to what is listed after the **C** command, and then relocks the unit.

7. Continue these same steps until a unique ID is assigned to each unit.

Note: Keep a list of the IDs; you will need this list in the section "Transferring Data from Multiple Data Collectors."

Changing a Data Collector's ID with a DOS Computer

To assign a unique ID when using a DOS computer, create a commands file, and then enter the name of the commands file on the **Download.exe** command line. **Download.exe** then uses the commands file to unlock the data collector and change the ID.

During normal operation, a data collector is given an ID of ten zeros. To change the ID, you must create a commands file consisting of the following three command lines:

1) an **I** command, followed by a space, followed by ten zeros to unlock the unit;
2) a **C** command, followed by a space, followed by the new ID for the unit;
3) and an **L** command to relock the unit.

I 0000000000	Sends an unlock command to the data collector. (Note: Ten zeros will unlock any data collector.)
C *new ID#*	Changes the unit's ID to the given ID. The ID can be any alphanumeric combination of up to ten characters in length.
L	Sends a **LOCK** command and puts the data collector to sleep. When the unit is awakened with a keypress, it immediately runs the current application.

For example, to change a unit's ID to **12345**:

1. Use a word processing program to create a commands file containing the following three lines:

   ```
   I 0000000000
   C 12345
   L
   ```

2. Save the file as a text file with a **.txt**, **.vdx**, or no extension and quit the program.

3. Place a data collector into the Base Station slot.

IMPORTANT: When you are sending a file to a data collector, it must be the ONLY data collector attached to the computer.

4. Enter the name of the commands file on the **Download.exe** command line.

5. **Download.exe** executes the commands listed in the file: it unlocks the unit, changes the ID as instructed by the **C** command (in this case, 12345), and then relocks the unit.

6. Remove the data collector from the Base Station slot and it is ready to use.

To continue changing other data collector's IDs:

1. Reopen the commands file.

2. Change the ID listed after the **C** command by selecting **12345** and typing in a new ID. (Note: There must be a space character between the **C** and the ID.)

3. Save and close the file.

4. Insert a data collector into the Base Station slot.

5. Reactivate the **Download.exe** command line containing the name of the commands file.

6. **Download.exe** executes the commands listed in the file: it unlocks the unit, changes its ID as instructed by the **C** command, and then relocks the unit.

7. Continue these same steps until you have assigned a unique ID to each of your units.

Note: Keep a list of the IDs you are using; you will use this list in the section "Transferring Data from Multiple Data Collectors."

Changing a Data Collector's ID with a Macintosh Computer

To assign an ID with a Macintosh computer, you must first create a commands file, and then click and drag the file onto the **Videx Download** icon. **Videx Download** uses the commands file to unlock the data collector and change the ID. (Note: The LaserLite Mx is not Macintosh compatible.)

During normal operation, a data collector is given an ID of ten zeros. To change the ID, create a commands file consisting of the following three command lines:

1) an **I** command, followed by a space, followed by ten zeros to unlock the data collector;

2) a **C** command, followed by a space, followed by the new ID for the data collector;

3) and an **L** command to relock the data collector.

For example, to change a data collector's ID to **12345**:

1. Use a word processing program to create a file containing the following three lines:

   ```
   I 0000000000
   C 12345
   L
   ```

2. Save the file as a text file with a **.txt**, **.vdx**, or no extension and quit the program.

3. Place a data collector into the Base Station slot.

IMPORTANT: When you are sending a file to a data collector, it must be the ONLY data collector attached to the computer.

4. Click and drag the commands file onto the **Videx Download** icon.

5. **Videx Download** executes the commands listed in the text file: it unlocks the unit, changes the ID to what is listed after the **C** command (in this case, 12345), and then relocks the unit.

6. Remove the data collector from the Base Station slot; it is now ready to use.

To continue changing other data collector's IDs:

1. Reopen the commands file.

2. Change the ID listed after the **C** command by selecting **12345** and typing in a new ID. (Note: There must be a space character between the **C** and the ID.)

3. Save and close the file.

4. Insert another data collector into the Base Station slot.

5. Click and drag the edited commands file to the **Videx Download** icon.

6. **Videx Download** executes the commands in the commands file: it unlocks the unit, changes its ID as instructed by the **C** command, and then relocks the unit.

7. Continue these same steps until you have assigned a unique ID to each data collector.

Note: Keep a list of the IDs; you will use this list in the following section: "Transferring Data from Multiple Data Collectors."

Transferring Data from Multiple Data Collectors

To transfer data from a group of data collectors in one transfer process, you must:

1. Create a commands text file that uses the **FOREVER**, **FOREACH**, or **FORALL** looping commands to transfer the data from the data collectors.
2. Insert all of the data collectors into their Base Stations.
3. If using Windows, click and drag the commands text file onto the **Vxcom** icon; if using DOS, enter the name of the commands text file on the **Download.exe** command line; or if using Macintosh (DuraTrax, LaserLite, or LaserLite Pro only), click and drag the commands text file onto the **Videx Download** icon.

The three looping commands provide transfer methods for three different situations. The **FOREVER** loop is useful if you have a computer dedicated to transferring the data from the data collectors. The **FOREACH** loop is useful if you are transferring data from only a portion of the data collectors. The **FORALL** loop is useful if all of the data collectors are required to be transferred during the process.

The following table describes the looping commands:

FOREACH	This loop will search for each data collector in the list of IDs. If it locates the data collector with the ID, it executes the commands within the loop once; if it does not locate the data collector with the ID, it goes on to the next ID. This loop attempts to execute commands within a loop once for each ID listed.
FORALL	This loop will look for each listed data collector's ID until it has located and transferred the data from each ID in the list. This loop attempts to execute commands within a loop indefinitely, until all listed IDs have been located.
FOREVER	For use with a computer that is dedicated to transferring data. This loop will transfer the data from any data collector it locates. This loop attempts to execute commands within a loop indefinitely.

Table 1-1 Looping Commands

FOREVER...END - Attempts to execute commands within the loop indefinitely.

Example:

```
FOREVER                 'Begin the loop.
     I 0000000000
     'Unlock any unit.
     M1 Transferring Data 'Put message on display.
     S                  'Transfer data from unit.
     Z                  'Clear data from unit.
     T                  'Set time by computer’s time.
     G                  'Restart the application.
END             'End loop, begin re-executing at top.
```

Note on the Unlock (I) command: No additional characters should be included on the same line as the **Unlock** (**I**) command. Any characters after the ID (for example, space characters or comments) will cause this command to fail. In the above example, the comment associated with the **Unlock** command is on a separate line.

FOREACH ["id file.ext"] or [<ID-1, ID-2, ...>] ... END - Attempts to execute the commands within the loop <u>once</u> for each ID passed. Requires an argument that may be either a list of IDs or the quoted name of a file that contains a list of IDs. The list may be either comma or space delimited. If a file is used, the IDs may also be separated by carriage return/line feed characters.

Example:

```
FOREACH ID1, ID2, ID3         'Begin the loop.
   I loop_id
'Unlock the unit using the special identifier. This
'identifier substitutes for all the IDs listed on the
'first line.
   M1 Transferring Data 'Put message on display.
   S                    'Transfer data from unit.
   Z                    'Clear data from unit.
   T                    'Set time by computer’s time.
   L            'Lock the unit and put it to sleep.
END             'End loop, begin re-executing at top.
```

Note on the loop_id argument: When the argument of the **Unlock** command is a specific ID (as in the previous **FOREVER** example), the program attempts to unlock the unit eight times at five-second intervals. This ensures that a unit is unlocked regardless of its sleep cycle (a unit sleeps for 25 seconds, then wakes for 5 seconds). If the program does not unlock a unit, the program returns an error and aborts. On the other hand, when the argument of the **Unlock** command is **loop_id** (as in the **FOREACH** example), the program only makes one attempt to unlock the unit. If it is successful, it performs the remaining commands in the loop; if not successful, it moves on to the next value for **loop_id** without aborting the program. Given the unit's sleep cycle, it is unlikely that one attempt will successfully unlock a unit. Therefore, it would generally not be reasonable to use a **FOREACH** loop by itself as in the above example. See pages 41–42 for examples that show the appropriate use of the **FOREACH** loop.

FORALL ["id file.txt"] or [<ID-1, ID-2, ...>] ... END - Attempts to execute the commands within the loop for each ID passed. **FORALL** continues to attempt the commands within the loop until all commands are successfully completed for all ID's. When all commands are completed for one ID, it is removed from the list and that ID isn't attempted again. Requires an argument that may be either a list of IDs or the quoted name of a file that contains a list of IDs. The list may be either comma or space delimited. If a file is used, the IDs may also be separated by carriage return/line feed characters.

Example:

```
FORALL "IDs.txt"        'Begin the loop.
   I loop_id
'Unlock the unit using special identifier. This
'identifier substitutes for all IDs in the IDs.txt
'file.
   M1 Transferring Data 'Put message on display.
   S                    'Transfer data from unit.
   Z                    'Clear data from unit.
   T                    'Set time by computer's time.
   L           'Lock unit and put it to sleep.
END            'End loop, begin re-executing at top.
```

In order to clarify the functioning of these different looping structures, it is helpful to consider four different scenarios for transferring data from multiple units.

Scenario #1: The user has multiple data collectors to transfer, but only one at a time will be put in the Base Station or in front of the JetEye. A computer is dedicated to transferring data from the data collectors, as they become available. In this situation, the units do not need to be assigned individual IDs and a **FOREVER** loop can be used:

```
FOREVER
I 0000000000
M1 Transferring Data
S
Z
T
L
END
```

This loop continues indefinitely, unlocking any data collector that is available. It is important that only one unit at a time is in the Base Station or in front of the JetEye when using this looping structure. This loop will unlock any data collector; if more than one data collector is unlocked at a time, communications will fail.

Scenario #2: The user has multiple data collectors to transfer. Some or all of them may be in the Base Station at any time. Each unit must have its data transferred exactly one time. A computer is dedicated to transferring the data from the units as they become available. In this situation, each unit must be assigned a unique ID and the **FORALL** loop can be used:

```
FORALL "IDs.txt"
       I loop_id
       M1 Transferring Data
       S
       Z
       T
       L
END
```

This loop continues until the data from each unit in the **IDs.txt** file has been transferred once. Multiple units can be in the Base Station at one time because only one unit will be unlocked at a time.

Scenario #3: The user has multiple data collectors to transfer. Some or all of them may be in the Base Station at any time. The data from each unit is to be transferred when the unit is placed in the Base Station; some of the units may have their data transferred more than one time. A computer is dedicated to transferring the units as they become available. In this situation, each unit must be assigned a unique ID and a **FOREACH** loop nested inside a **FOREVER** loop can be used:

```
FOREVER
      FOREACH "IDs.txt"
            I loop_id
            M1 Transferring Data
            S
            Z
            T
            L
      END
END
```

This loop continues indefinitely, continually attempting to unlock any unit contained in the **IDs.txt** file. Unlike the **FORALL** loop in Scenario #2, this loop continues attempting to transfer the data from the unit even after it has been successfully transferred one time.

Scenario #4: The user has multiple data collectors to transfer; all of the available units are in their Base Stations. However, some of the units (listed in the **IDs.txt** file) are not available to transfer. If the computer is unable to unlock a particular unit, the computer stops trying to unlock that unit. In this situation, each unit must have a unique ID and a **FOREACH** loop can be used:

```
FOREACH "IDs.txt"
      I loop_id
      M1 Transferring Data
      S
      Z
      T
      L
END
```

As was explained in the earlier discussion of the **FOREACH** loop, when the argument of the **Unlock** command is **loop_id**, the program only makes one attempt to unlock the unit. Given the unit's sleep cycle, it is unlikely that one attempt will successfully unlock a unit. However, it is possible to ensure that each unit is awake when the program attempts to unlock it. This is done by placing six unused IDs at the beginning of the **IDs.txt** file (Figure 1-2).

```
Dummy1
Dummy2
Dummy3
Dummy4
Dummy5
Dummy6
ValidID1
ValidID2
ValidID3
ValidID4
```

Figure 1-2 Sample IDs.txt File

If a unit is awake and receives an **Unlock** command with the wrong ID, it will stay awake as long as it detects activity on the serial port. By placing six unused IDs at the beginning of the **IDs.txt** file, it is possible to ensure that each unit receives at least one **Unlock** command with an invalid ID and then stays awake. Then each unit is awake when the **FOREACH** loop attempts to unlock it and transfer its data. The program only attempts to unlock each unit one time; if a unit is not present, the program moves on to the next ID. The program quits after attempting one time to unlock and transfer the data from each unit.

Chapter 2

Monitor Program

This chapter contains information on:

- The monitor program.
- Resetting the data collector to monitor mode.
- The monitor commands.
- Recovering data.
- Loading to LaserLite Pro or LaserLite Mx flash memory.

Monitor Program Overview

The monitor program is a boot loader named **VX1 Monitor** that resides in the data collector's ROM. **VX1 Monitor** allows updating the RAM from the serial port and accepts system software in Intel HEX ASCII file format (uploaded at 9600 baud, no parity, 8 data bits, 1 stop bit).

Resetting the Data Collector to Monitor Mode

In the unlikely event that your data collector locks up and does not recover after two minutes, you must manually reset the unit to monitor mode and resend the operating system software along with an application to the data collector.

To manually reset the data collector to monitor mode:

1) Remove the battery end cap.
2) Hold the scan button down while replacing the battery end cap.
3) The unit will beep three times, indicating a successful reset.
4) The DuraTrax and LaserLite will show **VX1 Monitor 1.07** on the first line, the LaserLite Pro will show **VX1 Monitor 1.11** or **1.34** on the first line, and the LaserLite Mx will show **VX1 Monitor 1.33** or **1.34** on the first line. All four units will show **Ready, bps=9600** on the second line of the display.
5) Use the appropriate communications program (**Vxcom** for Windows, **Download** for DOS, or **Videx Download** for Macintosh) to resend the operating system software. (Note: You must use **Vxcom** or **Download** if communicating with a LaserLite Mx with a memory card.)

See Chapter 1 for information on using the communications programs. See Chapter 3 for information on the operating system software.

Monitor Program Signatures

DuraTrax and LaserLite:
VX1 Monitor 1.07
ROM checksum: 59FF

LaserLite Pro:
VX1 Monitor 1.11 (or 1.34)
ROM checksum: 52AC

LaserLite Mx:
VX1 Monitor 1.33 (or 1.34)
ROM checksum: 52AC

Monitor Commands

The monitor program prompts the host for a command by sending an asterisk character (*) out of the serial port preceded by a \r\n (carriage return and line feed). Following is a list of the commands and their descriptions.

B Read battery level
Returns a hex value directly from the A to D chip. Range is 00–FF.

To convert to voltage:

1) Convert the hexadecimal value to a decimal value.
2) Multiply the decimal value by 0.02898.

C Read config
Returns the software version.

D ssss eeee Dump hex range
Returns RAM contents from beginning of range (ssss) to end of range (eeee) in Intel HEX packets. This command may be used to dump the contents of RAM from the data collector to a file on the computer.
D 0000 FFFF dumps the first bank of memory (BANK 0).

G aaaa **Go to address**
Transfers control from the monitor program to the system software. The Videx RAM resident software always uses 7FFB as the entry point. To start RAM resident system software, enter **G 7FFB**. To restart the monitor program send **G 0**.

H **Help screen**
Displays a brief list of commands supported by the monitor program (VX1 Monitor 1.07 only).

L **Load hex**
Puts the monitor into a load mode and prompts the host with a question mark character for the first line of the bootstrap hexadecimal file. After the (?) prompt has appeared, the only valid inputs to the monitor are a properly formatted Intel Hex line of data or a blank line.

After the monitor has processed each valid line of input, it responds with another load prompt (?). If the monitor receives an improperly formatted line or invalid information, it drops back into the monitor command mode and issues a monitor prompt (*).

It is possible for the host to recover from such an error by re-entering the load mode and re-issuing the incorrectly received line.

When the monitor receives the Intel Hex End of File record (:00000001FF\r\n), the unit displays **loading finished**, **ready**,**bps=9600**.

If the monitor receives an erroneous line, it returns to the command line mode and displays **loading failed**, **ready**, **bps=9600**.

If the monitor does not receive any data for 30 seconds, it returns to the command line mode and displays **loading time-out**, **ready**, **bps=9600**.

Pn **Read port**
Read the I/O page of the memory bank. Returns a 1 byte value.

Pn=dd **Write port**
Sets the I/O page of the memory bank to address. To set P1 to address the desired RAM bank, use the following pattern:

32K memory bank number	Modify the value of P1
Bank 0	Clear bits 2, 3
Bank 1	Set bit 2, clear bit 3
Bank 2	Clear bit 2, set bit 3
Bank 3	Set bits 2, 3

(See the system diagrams on pages 48–50.)

R aaaa **Read external**
Returns the contents of RAM memory from the given address. Displayed on the computer screen in both hexadecimal and decimal ASCII.

RI **Read internal**
Returns the contents of the processor memory.

S **Sleep**
Puts the system in low-power mode. Returns "bye."
The data collector may be reawakened by pressing any button or by sliding the lock switch from the OFF to the ON position.

T **Read RTC**
Returns the current date and time stored in the real-time clock. Format is MMDDYYHHMMSS. (VX1 Monitor 1.07 and 1.11 only.)

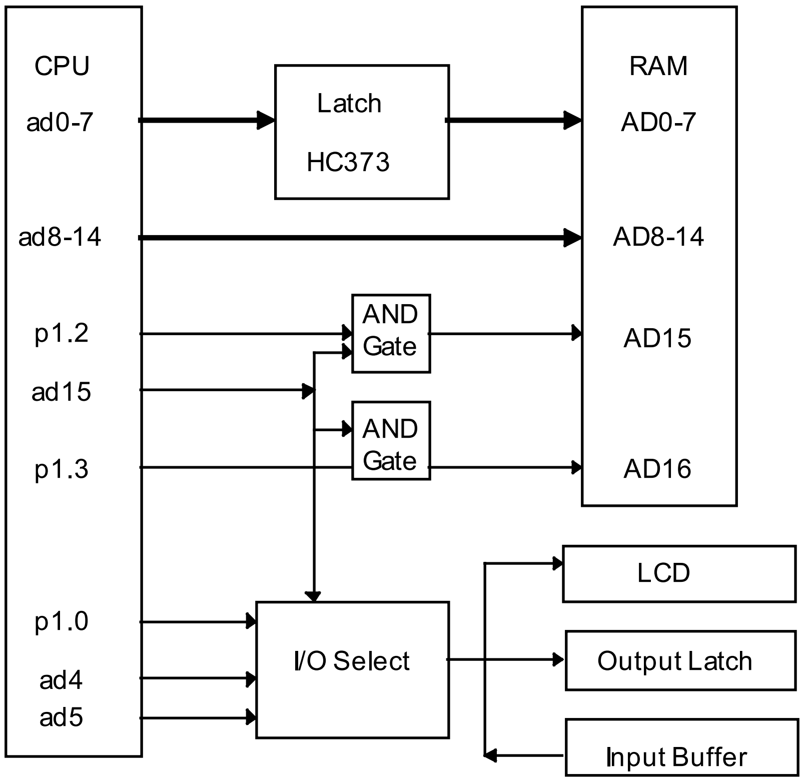
T mm dd yy hh mm ss **Write RTC**
Writes a new value to the real-time clock. If it receives invalid input, the monitor returns an "out of range" error message. (VX1 Monitor 1.07 and 1.11 only.)

V ssss eeee **Verify range**
Calculates and returns a CRC checksum for a given range of memory.

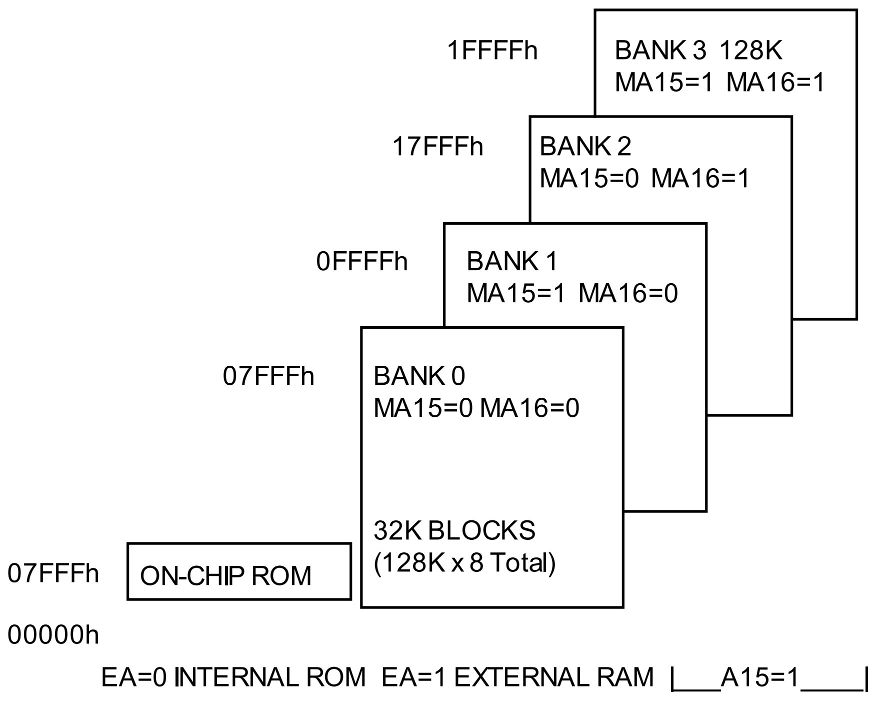
W aaaa dd **Write external**
Enables writing a byte value to a given memory address in system memory.

WI aa dd **Write internal**
Enables writing a byte value to a given memory address in processor memory.

System Block Diagram

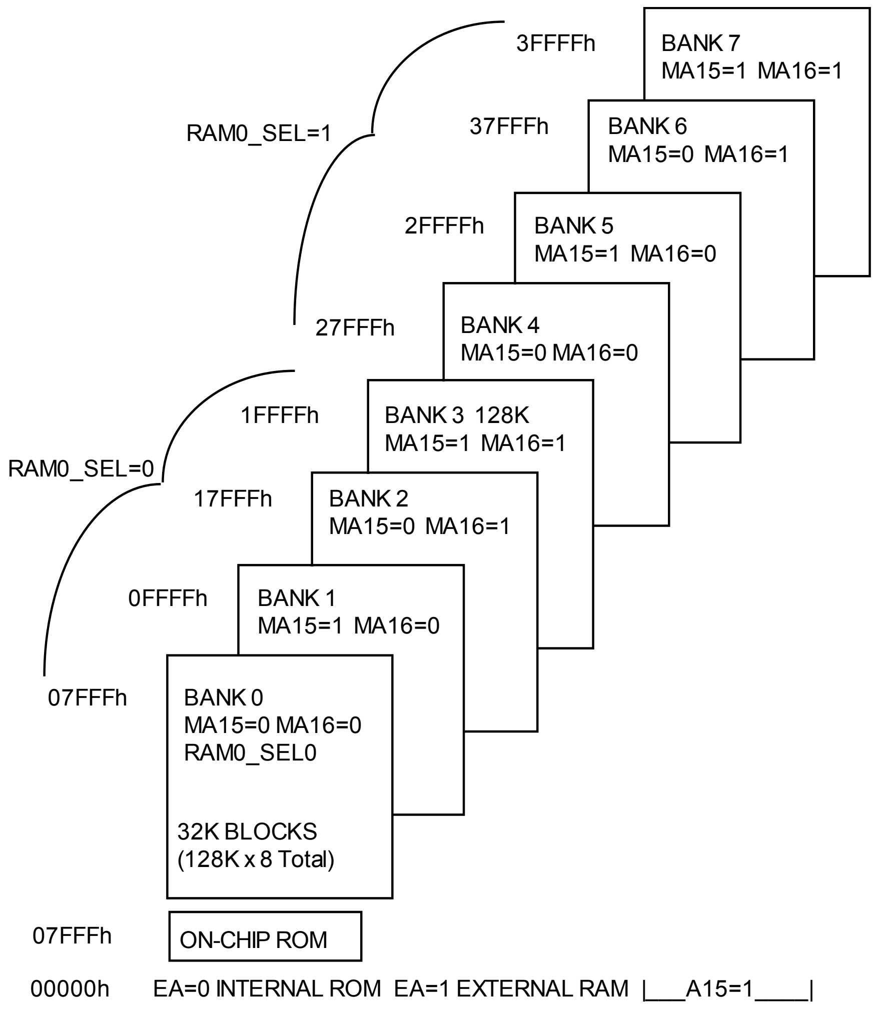


System Memory Map Diagram



DuraTrax/LaserLite System Memory Map Diagram

System Memory Map Diagram (continued)



LaserLite Pro/LaserLite Mx System Memory Map Diagram

Dumping Memory to Recover Data

In the event of an operating system lock-up, memory may be dumped from the DuraTrax, LaserLite, or LaserLite Pro for the purpose of data recovery prior to loading a new operating system and application. It should not be necessary to dump data from a LaserLite Mx since its data resides on the memory card independently from system memory.

Caution must be exercised to set and reset the bits to the correct values or the data collector may not operate properly.

Components required:

- A computer terminal program with a "capture" mode (i.e., Procomm).
- A Videx Base Station with serial cable attached to a computer.
- Software to convert 32 character per line Intel HEXASCII to binary. The public domain program BINTEL.COM is suitable.
- A calculator or computer program like Windows CALC.EXE that can convert hexadecimal values to binary values.

You may use the **R** *xxxx xxxx* to address up to 64K of memory with addresses ranging from 0000 to FFFF. Remember that 0000 to 7FFF is always RAM 0, BANK 0 (see the system memory map diagrams on the previous pages).

To select each bank of memory (0–3) in RAM0_SEL = 0, RAM0_SEL = 1, or FLASH, use the following values:

BANK0	MA15=0	MA16=0
BANK1	MA15=1	MA16=0
BANK2	MA15=0	MA16=1
BANK3	MA15=1	MA16=1

To select RAM0_SEL or FLASH, use these values:

RAM0_SEL=0	8070=F8
RAM0_SEL=1	8070=F9
FLASH	8070=FC

The general steps are as follows:

1. Read P1; it returns a hexadecimal value. Translate it to a binary value. For example, if P1=94, the binary value of 94 hex is 10010100.

Bit Number	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Example Value	1	0	0	1	0	1	0	0
					MA16	MA15		Enables writing to the latch.

2. If needed, set the value of bit 0 to 1 to enable writing to the latch. Use the following commands to write to the latch:

```
RAM0_SEL=0  W 8070=F8
RAM0_SEL=1  W 8070=F9
FLASH       W 8070=FC
```

3. If you set bit 0 in Step 2, clear it.

4. Dump the memory (D *0000 FFFF*).

5. Repeat for the next memory bank as needed.

6. Convert the hexadecimal files to binary files and combine as needed.

7. Data stored in the data collector is printable ASCII. You may use a text editor to view the binary files and retrieve the data.

Special Notes on LaserLite Pro Monitor 1.11

Loading from Flash Memory

At startup, the monitor program in the LaserLite Pro and the LaserLite Mx reads signature bytes to determine whether to copy the contents of flash memory to RAM. The signature bytes are located as follows:

In RAM:	7FF9 and 7FFA	Bank 0
In Flash:	FFF9 and FFFA	Bank 0

Signature byte values:

Location	Value	Monitor program actions at startup	What writes this value and when is it written?
FFF9–FFFA (flash)	5A–A5	Valid image file is stored in flash.	Monitor program after validating the image file is loaded in the flash.
7FF9–7FFA (RAM)	5A–A5	Valid image file is stored in RAM. Do not load image file from flash. Automatically start the operating system at 7FFB after monitor program startup.	Monitor program when it copies the image file from flash.
7FF9–7FFA	3C–C3	Valid image file is stored in RAM. Do not load image file from flash. Do <u>not</u> automatically start the operating system after monitor program startup.	The operating system when it quits execution and returns to the monitor program.
7FF9–7FFA	any other values	No valid image file is stored in RAM. Check flash. If valid image file is stored in flash, copy contents of flash to RAM and execute at 7FFB.	All other values indicate that RAM contents are lost or corrupted. This could occur if batteries were removed from the system for several weeks and the backup battery drains.

Startup Model for VX1 Monitor 1.33 and 1.34

Each time the LaserLite Pro and LaserLite Mx awaken from sleep or when they are reset, the system starts with the monitor program. At that time, the monitor program evaluates seven conditions. Based on their values, it selects an appropriate path of execution.

These are the seven conditions it evaluates and the bit values it sets:

1. Reads the bytes at 7FF9 and 7FFA. If these bytes evaluate to 3C C3 then the monitor program starts and does not read any of the other values.

3C C3 at 7FF9?	Bit 0
Yes = 0000001	
No = 0000000	

2. Reads internal memory to match against a known valid pair of bytes (5A A5 for LaserLite Pro, 3C C3 for LaserLite Mx). If it matches, this indicates that the data collector went through a normal shutdown and saved the state of the processor. Otherwise it indicates an abnormal shutdown such as a power loss during operations.

Valid Internal Signature	Bit 1
Yes = 0000010	
No = 0000000	

3. Reads external memory to match against a known valid pair of bytes (5A A5). If it matches, this indicates that the data collector has a valid operating system loaded into memory. Otherwise, it indicates no operating system is loaded. This could occur if both the main batteries and the backup batteries were allowed to completely drain.

Valid External Signature	Bit 2
Yes = 0000100	
No = 0000000	

4. Checks to see if there is a memory card module, if a memory card is present, if it has a recognizable format, and if the memory card has a system boot file. The monitor sets flags regarding these “memory card” checks that may be read by the LaserLite Mx operating system at startup.

Card_is_bootable	Bit 3
Yes = 0001000	
No = 0000000	

5. Reads flash memory to match against a known valid pair of bytes (5A A5). If it matches, this indicates that the data collector has a valid operating system and application loaded into flash.

Application in Flash	Bit 4
Yes = 0010000	
No = 0000000	

6. Reads the state of the scan button.

Scan button down	Bit 5
Yes = 0100000	
No = 0000000	

7. If there is a memory card, it reads the ID of the memory card and compares it against a value stored in external memory at 7F D1. The monitor uses this address to store the most recent ID of the memory card from which the LaserLite Mx booted.

Card ID <> value at 7DF1	Bit 6
Yes = 1000000	
No = 0000000	

The following table uses a bitmap model to indicate the path of execution it will select.

IF Bit 0 = 1	Go to Monitor Mode
IF Bit 6 = 1 and Bit 3 = 1	Boot from SSFDC

IF 0000010	Go to Monitor Mode
IF 0000100	Set power_fail flag then start OS
IF 0000110	Start OS
IF 0001000	Boot from SSFDC
IF 0001010	Boot from SSFDC
IF 0001100	Set power_fail flag then start OS
IF 0001110	Start OS
IF 0010000	Boot from Flash
IF 0010010	Boot from Flash
IF 0010100	Set power_fail flag then start OS
IF 0010110	Start OS
IF 0011000	Boot from SSFDC
IF 0011010	Boot from SSFDC
IF 0011100	Set power_fail flag then start OS
IF 0011110	Start OS
IF 0100000	Go to Monitor Mode
IF 0100010	Go to Monitor Mode
IF 0100100	Set power_fail flag then start OS
IF 0100110	Start OS
IF 0101000	Boot from SSFDC
IF 0101010	Boot from SSFDC
IF 0101100	Set power_fail flag then start OS
IF 0101110	Start OS
IF 0110000	Go to Monitor Mode
IF 0110010	Go to Monitor Mode

IF 0110100	Set power_fail flag then start OS
IF 0110110	Start OS
IF 0111000	Boot from SSFDC
IF 0111010	Boot from SSFDC
IF 0111100	Set power_fail flag then start OS
IF 0111110	Start OS
IF 1000000	Go to Monitor Mode
IF 1000010	Go to Monitor Mode
IF 1000100	Set power_fail flag then start OS
IF 1000110	Start OS
IF 1010000	Boot from Flash
IF 1010010	Boot from Flash
IF 1010100	Set power_fail flag then start OS
IF 1010110	Start OS
IF 1100000	Go to Monitor Mode
IF 1100010	Go to Monitor Mode
IF 1100100	Set power_fail flag then start OS
IF 1100110	Start OS
IF 1110000	Go to Monitor Mode
IF 1110010	Go to Monitor Mode
IF 1110100	Set power_fail flag then start OS
IF 1110110	Start OS

Notes:

Chapter 3

Operating System Software

This chapter contains information on:

- The operating system software.
- The operating system software commands.
- The **Default.s** application.
- LaserLite Mx memory card operating system commands.

Operating System Software Overview

The operating system software (**OS**) must be installed on the data collector before it can gather and store data. The filename of the operating system software will always end with a **.OS** extension. The exact filename of the **OS** will change as new versions are released; however, the **OS** for the DuraTrax will always begin with **Trax**, the **OS** for the LaserLite will always begin with **Lite**, the **OS** for the LaserLite Mx will always begin with **Lmx**, and the **OS** for the LaserLite Pro will always begin a **Pro**. (Note: Double-check the actual name of your operating system files.)

The **OS** consists mostly of a virtual machine that executes an application. An application is a special binary machine language compiled from BASIC code that includes instructions to gather events, manipulate integers and strings, manage files, talk to the data collector, and so on. The virtual machine provides all of the functionality available to developers of DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx software.

In addition to the virtual machine, the **OS** file also presents a command line interface (using the serial port for **stdin** and **stdout**), with commands to send and receive files, get the battery level, get and set the clock, change the data collector's ID, and so on.

When the **END** instruction is executed by the virtual machine, the **OS** returns to the command line. When the **G** or **L** commands are issued to the command line, the **OS** executes the virtual machine. (The **G** command is automatically executed by the **OS** if no characters are received for 60 seconds, or if an **Unlock** command is received with the wrong ID.)

The virtual machine executes the current application. There can be only one application at a time installed with the **OS**.

Operating System Command Line

After the BASIC program ends, the operating system returns to a command line, analogous to the DOS command line. It uses the serial port for standard input and output, so you can communicate with the data collector using a Videx Base Station and standard communications software.

When the operating system first returns to the command line, it displays the version of the operating system software on one line, and the data collector's ID on the next line. It then prompts the user with the standard ">" prompt.

If you are using a terminal program, you can type a command at the prompt terminated with a carriage return character. The operating system software will respond to the command, and then display a prompt again. If the command was completed successfully, it displays a ">" prompt. If the command couldn't be completed for some reason, it displays a "!" prompt.

Standard command syntax is: **<command> [argument] <cr>**
All operating systems will ignore the linefeed character after a carriage return, except with the **I** command.

Any command that causes the size of the data file to be reset to 0 must be sent three times in quick succession; this includes the **Default Application**, **Receive**, **Clear Data**, and the **Load from Flash** commands. After the first and second command, a "+" prompt is displayed, to indicate that it is waiting for more. After the third command, the normal ">" prompt is displayed if the command is successful. If the second or third commands aren't sent quickly enough, then the error prompt "!" is displayed. (The second and third commands must be received within two to three seconds after the first command.)

The command line has a 60-second timeout. If no characters are received for 60 seconds, it automatically executes the **G** command.

Operating System Commands

All commands consist of a single command letter. The command letter is not case sensitive. For some commands, an argument follows the command letter. Any spaces between the command and the first letter of the argument are ignored. (None are required.) The argument extends from the first non-space character to the carriage return that ends the line. Only the first 31 characters of the argument are significant; the rest are discarded.

Battery

The **Battery** command is a **B** with no arguments. It responds with the current battery voltage (for example, “5.65”).

Set ID

The **Set ID** command is a **C** with one optional argument. If the argument is supplied, the unit’s ID is changed to the first ten characters of the argument. With or without the argument, it responds with the unit’s ID.

Default Application*

The **Default Application** command is a **D** with no arguments. <u>Since this command clears data it must be sent three times.</u> In operating systems prior to OS122, this sets the current application to be the default application built into the operating system software. Any data is destroyed, and any application that was previously installed is also destroyed. A listing of the default application is contained in the disk accompanying this software package. The filename of the default application for DuraTrax, LaserLite, and LaserLite Pro is **Default.s**. The filename of the default application for LaserLite Mx is **MX-DEMO.s**. See pages 66–73 for more information on the default application.

*For operating systems 122 and later, this command behaves identically to the **Z** (clear data) command.

Load from Flash *(OS versions 1.22 or later, LaserLite Pro and LaserLite Mx only)*

The **Flash** command is an **F** with no arguments. Since this command clears data on the LaserLite Pro and the LaserLite Mx, it must be sent three times. In operating systems 1.22 or later, this command instructs the monitor program to check the unit's flash memory for a valid image file (***.IMG**) and, if found, load that image file into RAM. Any data is destroyed, and any application previously in the LaserLite Pro or LaserLite Mx is overwritten.

If this command is executed on a DuraTrax or LaserLite (OS versions 1.22 or later) it behaves exactly like the **Q** (quit) command. It does not destroy data on the DuraTrax or LaserLite.

Go

The **Go** command is a **G** with no arguments. This causes the current BASIC application to be executed.

Unlock

The **Unlock** command is an **I** (for identify) with one argument. The argument is either the unit's ID or ten zeros. Ten zeros will unlock any Videx data collector. The **I** (**Unlock)** command must be terminated with only a carriage return, not a carriage return/line feed combination. Both the BASIC application and the command line respond to this command.

If an **I** (**Unlock**) command is sent to the BASIC application, an exit event is returned the next time **INPUTEVT** is called. In this case, there must be a delay of at least 500 ms between sending the command and calling **INPUTEVT** without any other characters being sent. The BASIC program should typically respond to an exit event by ending the program.

If an **Unlock** command is used without an argument, it responds with the system version and unit ID. (This is the same information that is displayed when the command line is first entered.) If an **I** is received, but the argument doesn't match ten zeros or the unit ID or if it is terminated with a carriage return and linefeed, then the data collector immediately executes the **G** command. This prevents two units from being unlocked at the same time.

The **I** command must be terminated with only a carriage return, not a carriage return-line feed pair.

Lock

The **Lock** command is an **L** with no arguments. It causes the data collector to go to sleep immediately. When the unit is awakened, it executes the BASIC application.

Message

The **Message** command is an **M**, followed immediately by either a 1 for the top line of the display or a 2 for the bottom line, followed by one argument. The argument is written to the display, starting at the beginning of the indicated line.

Pass to Memory Card *(LaserLite Mx only)*

The **Pass to Memory Card** command is a **P** followed by a command string. The command strings are the LaserLite Mx memory card command set; these are the same commands that are used in the Videx BASIC **CARDCMD** statement's *command_str$* parameter. See pages 105–129 for information on the **Pass to Memory Card** command strings.

Quit Operating System

The **Quit Operating System** command is a **Q** with no parameters. It causes the operating system in the data collector to stop executing and restarts the monitor program. This command neither destroys data nor removes the application. To restart the operating system, enter **G 7FFB** at the monitor command line.

Receive

The **Receive** command is an **R** with no arguments. Since this command clears data it must be sent three times. It causes the data collector to start receiving a new application using the YModem protocol. Any data contained in the data collector is destroyed.

The YModem protocol is XModem CRC, with optional 1K packets, and batch extensions. (Some communications programs claim to implement YModem, when they really just implement one or two extensions to XModem.)

If the application is not successfully received, the **D** command is executed and the default application, if present, is installed. If the application is successfully received, it is installed as the current application.

Send

The **Send** command is an **S** with no arguments. It causes the data collector to start sending the data file using the YModem protocol. The name of the data file is the same as the name last used by the BASIC application to open it.

Time

The **Time** command is a **T** with one optional argument. If the argument is supplied, it must be in the form: MM DD YY hh mm ss (Note: The spaces are optional.) The data collector's clock is changed to the given time. With or without the argument, it responds with the current time in the form: MM-DD-YYYY hh:mm:ss.

Reset Memory Card

The **Reset Memory Card** command is an **X** with no arguments. It restarts the memory card system if it stops because the card was removed.

Listen for Response from Memory Card

The **Listen for Response from Memory Card** command is a **Y** with no arguments. It watches the data stream coming from the memory card until it receives a carriage return. Then it captures any incoming data until it receives a second carriage return. It times out in two seconds in the absence of a carriage return. This special command is used for monitoring progress when formatting a memory card. See Appendix B for information on formatting the memory card.

Clear Data

The **Clear Data** (zap) command is a **Z** with no arguments. Since this command clears data it must be sent three times. It causes the data file's size to be set to 0. Any data is destroyed.

Default.s Application

The DuraTrax, LaserLite, and LaserLite Pro are shipped with an application called **Default.s** installed. The **Default.s** application allows entry of any data and transfers the data as a formatted file that includes the entry and the date and time of each entry. **Vxcom** or **Download** will automatically load the **Default.s** application into a DuraTrax, LaserLite, or LaserLite Pro if another application is not listed on the command line along with the operating system. The **Default.s** application and the **Default.b** source code are located on the disk accompanying this software package. The following tables describe the variables and sounds used by the **Default.s** application.

Default Application Variables

Variable	Description
bar$, bar2$	Temporary string variables used at various points in the program.
data$	64-character string containing data that was read by unit. Initialized to null string. Obtains value from **INPUTEVT** statement.
delimiter$	Delimiter between lines of the data file. Set to <cr><lf>.
device%	Parameter returned by **INPUTEVT** indicating type of device that entered the input data. Returns a 1 for keyboard, 2 for laser, 12 for contact scanner, 48 for touch button, or 64 for application generated (not possible with **Default.B**). Program does nothing with **device%** except for checking whether user has pressed the LaserLite Pro's ENT (Enter) key without entering any data — in this case, entry is ignored and unit sounds a single low beep.
display$	Character string containing characters displayed on unit. Initialized to null string.
foo%	Temporary parameter used at various points in the program.
full_display$	String containing complete input data. Used by scroll left and scroll right routines.
high%	Current size of data file in high bytes (multiples of 32768). Used when deleting the most recent data entered.

Variable	Description
info_index%	Index used when cycling through display of various system parameters.
low%	Current size of data file in low bytes (modulo 32768). Used when deleting the most recent data entered.
mode%	Flag indicating mode. Set to 0 for normal, 1 for scrolling up, and 2 for scrolling down.
mode_dn%	Setting for scrolling down the file or scanning. Set to 2.
mode_norm%	Setting for having scrolled up to the top or down to the bottom of the file. Set to 0.
mode_up%	Setting for scrolling up through the file. Set to 1.
nbytes%	Number of bytes to delete from the data file when deleting the most recent data entered.
ri%	Loop index used when alerting user to error opening data file.
running%	Flag indicating whether to keep running program. Set to 1 (or true%) while running; set to 0 (or false%) when unit receives **I** (**Unlock**) command at serial port.
shift%	Index used to control display during scroll left and scroll right routines.
symbol%	Parameter returned by **INPUTEVT** indicating bar code symbology of input data. Returns a 1 for Code 39, 2 for UPC-A or UPC-E, 4 for Interleaved 2 of 5, 8 for Codabar, 16 for EAN, 64 for UCC Code 128, or 128 for Code 128. The default application does nothing with this value.

Variable	Description
type%	Parameter returned by **INPUTEVT** indicating type of input event. Returns a 1 for input, 2 for exit, 3 for scanning a 5-space bar code that will delete the last input), 4 for scroll up key, 5 for scroll down key, 6 for power off switch, 7 for unexpected power loss, 8 for ESC key, 9 for scan button pressed and released, 10 for no memory card module found, 11 for no memory card found, 12 for unknown memory card found, 13 memory card ID has changed, 14 for memory card processor interrupted, 18 for scroll left key, 19 for scroll right key, 20 for MEM key, 21 for BAT key, 22 for f1 key, 23 for f2 key, 24 for f3 key, 25 for f4 key, or 26 for f5 key. (The LaserLite Mx recognizes all of these events, the LaserLite Pro recognizes events 1–9 and 18–26, and the LaserLite and DuraTrax recognize events 1–7 and 9.)
voltage%	Current battery voltage.
volt1%, volt2%	Battery voltage thresholds that trigger low voltage warnings.

Default Application Sounds

Description of Sound	Code	Event
Single click	sound 1397, 10	A key was pressed.
Single high beep	sound 1446, 250	Good input accepted (good beep).
Single low beep	sound 698, 250	Enter (ENT) key pressed without any data entered.
Two beeps: high, medium	sound 3620, 150 sound 2349, 250	Most recent data deleted.
Two beeps: medium, high	sound 2349, 150 sound 3620, 250	Confirm that most recent data is to be deleted.
Three beeps: high, low, high	sound 2793, 250 sound 2637, 250 sound 2793, 250	Begin default application (startup beep).
Three beeps: high, medium, low	sound 1760, 250 sound 1568, 250 sound 1397, 250	Exit default application (exit beep).

LaserLite Mx MX-DEMO Application

LaserLite Mx is shipped with the **MX-DEMO** application installed. **MX-DEMO** allows entry of any data and stores each scan with date and time as a single, tab-delimited record. **MX-DEMO** is also on the 2MB and 4MB memory cards sold by Videx as part of the CPU boot file. The application (**MX-DEMO.S**) and its source code (**MX-DEMO.B**) are included with the disk accompanying the LaserLite Mx software package.

The following table lists the memory variables and sounds used by the **MX-DEMO** application.

MX-DEMO Application Variables

Variable	Description
bar$, bar2$	Temporary string variables used at various points in the program.
buffer_var$	256-character string used capture the version string from the card in the init_card routine.
cardreturn$	256-character string for capturing data sent by the memory card processor using the **CARDSTATUS()** function.
crdcmd$	256-character string for building a command with parameters to pass to **CARDCMD**.
cstatus$	String variable dimensioned for capturing memory card status string. Not implemented with this version of **MX-DEMO**.
chandle$	File handle number of the currently open data file on the memory card. Used as a shorthand way to switch between files.
card_space$	String to hold number of free Kbytes on memory card. Used in evt_mem routine.
data$	64-character string containing data that was read by unit. Initialized to null string. Obtains value from **INPUTEVT** statement.
device%	Parameter returned by **INPUTEVT** indicating type of device that entered the input data. Returns a 1 for keyboard, 2 for laser, 12 for contact scanner, 48 for touch button, or 64 for application generated (not possible with **MX-DEMO.b**). Program does nothing with **device%** except for checking whether user has pressed the ENT (Enter) key without entering any data — in this case, entry is ignored and unit sounds a low beep.

Variable	Description
dhandle$	File handle number of the data file on the memory card.
display$	Character string containing characters displayed on unit. Initialized to null string.
foo%	Temporary parameter used at various points in the program.
full_display$	String containing complete input data. Used by *scroll left* and *scroll right* routines.
info_index%	Index used when cycling through display of various system parameters.
loopcount%	Integer to keep track of the next number to generate in the *rwloop* routine.
looping%	Flag indicating whether to continue in the F3, *number seek* routine.
mode%	Flag indicating mode. Set to 0 for normal, 1 for scrolling up, and 2 for scrolling down.
mode_dn%	Setting for scrolling down the file or scanning. Set to 2.
mode_norm%	Setting for having scrolled up to the top or down to the bottom of the file. Set to 0.
mode_up%	Setting for scrolling up through the file. Set to 1.
nbytes%	Number of bytes to delete from the data file when deleting the most recent data entered.
nodupes%	A toggle that indicates to the input routine whether to allow duplicate scans.
rdate$	Variable to hold and display the date field from seek record in F3, *number seek* routine.
record$	64-character string built to create a record that will be stored on the memory card. Initialized to null string. Built from data$ + <tab> + time + <tab> + date.
ri%	Loop index used when alerting user to error opening data file.
rtime$	Variable to hold and display the time field from seek record in F3, *number seek* routine.

Variable	Description
running%	Flag indicating whether to keep running program. Set to 1 (or true%) while running; set to 0 (or false%) when unit receives **I** (**Unlock**) command at serial port.
seekprompt$	String to hold the prompt for the F3, seek a number routine.
shift%	Index used to control display during *scroll left* and *scroll right* routines.
symbol%	Parameter returned by **INPUTEVT** indicating bar code symbology of input data. Returns a 1 for Code 39, 2 for UPC-A or UPC-E, 4 for Interleaved 2 of 5, 8 for Codabar, 16 for EAN, 64 for UCC Code 128, or 128 for Code 128. The default application does nothing with this value.
timewritten$	Time and date string read back from memory card record in the *rwloop* routine.
total_files$	String to hold the count of the total number of files on the memory card.
type%	Parameter returned by **INPUTEVT** indicating type of input event. Returns a 1 for input, 2 for exit, 3 for scanning a 5-space bar code that will delete the last input), 4 for scroll up key, 5 for scroll down key, 6 for power off switch, 7 for unexpected power loss, 8 for ESC key, 9 for scan button pressed and released, 10 for no memory card module found, 11 for no memory card found, 12 for unknown memory card found, 13 memory card ID has changed, 14 for memory card processor interrupted, 18 for scroll left key, 19 for scroll right key, 20 for MEM key, 21 for BAT key, 22 for f1 key, 23 for f2 key, 24 for f3 key, 25 for f4 key, or 26 for f5 key. (The LaserLite Mx recognizes all of these events, the LaserLite Pro recognizes events 1–9 and 18–26, and the LaserLite and DuraTrax recognize events 1–7 and 9.)
vertoggle%	Index used when cycling through display of operating system version number and build date when the F4 key is pressed.
voltage%	Current battery voltage.
volt1%, volt2%	Battery voltage thresholds that trigger low voltage warnings.

MX-DEMO Application Sounds

Description of Sound	**Code**	**Event**
Single click	sound 1397, 10	A key was pressed.
Single high beep	sound 1446, 250	Good input accepted (good beep).
Single low beep	sound 698, 250	Enter key pressed without any data entered.
Two beeps: high, medium (*deletetone*:)	sound 3620, 150 sound 2349, 250	Most recent data deleted or end *rwloop* routine.
Two beeps: medium, high (*questiontone*:)	sound 2349, 150 sound 3620, 250	Confirm that most recent data is to be deleted or begin *rwloop* routine.
Three beeps: high, low, high	sound 2793, 250 sound 2637, 250 sound 2793, 250	Begin application (startup beep).
Three beeps: high, medium, low	sound 1760, 250 sound 1568, 250 sound 1397, 250	Exit application (exit beep).
Two beeps: long medium, long low (*card_error_tone*:)	sound 2349, 300 sound 1885, 600	Memory card error.

Notes:

Chapter 4

Application Builder Source Template

This chapter contains information on:

- The Application Builder source code.
- Manipulating the source code.
- Descriptions of the code generation directives.
- Description of the **Describe.src** template.
- Description of the **Timewand.src** template.

Application Builder BASIC Source Code

Application Builder generates BASIC source code, which can be modified by the user after it is generated. The ability to edit this source code can be useful for applications with exceptional requirements that cannot be handled in the Application Builder interface.

Generate BASIC Source Code for Application

To generate the BASIC source code, use the **Export Binary** command in the Application Builder's **File** menu. This command saves the created application as a BASIC source file (***.B** file) and a compiled application file (***.S** file). Or, you can save only the BASIC source file by holding down the <Ctrl> key before accessing the **File** menu. The command appears as **Export Source Only** in the menu and only the BASIC source file (***.B** file) is saved.

Modifications can be made to the application's BASIC source file with any editor; however, if any changes are made to the source file it must be recompiled into an application with **Vxbasicw.exe** or **Vxbasic.exe**. See the *Videx BASIC Manual* for information on **Vxbasicw.exe** and **Vxbasic.exe**.

Application Builder uses an extremely flexible method to generate source code. An application is generated into source code by parsing a Source Template file. The Source Template file is copied directly into the source file, until it reaches a tilde (~) character. All text between ~'s is interpreted as code generation directives. When a code generation directive is reached, text that the directive requests is written out, and then the source file continues copying verbatim. (Note: The software uses **Describe.src** as the default Source Template file; see pages 89–101 for more information on the variables, subroutines, sounds, and plug-ins used by **Describe.src**.)

For example, if an application has six Input Handlers with ID numbers from 0 to 5 (0 is the ID for the “Begin” Handler), and the Source Template looks like this:

```
'*
'*    dimensions
'*

      dim visit_state%(~LASTSTEP~)
      dim visit_step%(~LASTSTEP~)
```

then the generated code would look like this:

```
'*
'*    dimensions
'*

      dim visit_state%(5)
      dim visit_step%(5)
```

Since the last Input Handler in the application has an ID of 5, the code generation directive **LASTSTEP** is replaced with the number 5. All the rest of the Source Template file is copied directly to the code.

Note: To determine the ID number of an Input Handler, start with the “Begin” Handler as ID 0, then count from top to bottom and from left to right. For example, Figures 4-1, 4-2, and 4-3 show three different applications consisting of six Input Handlers and the corresponding ID number for each Input Handler.

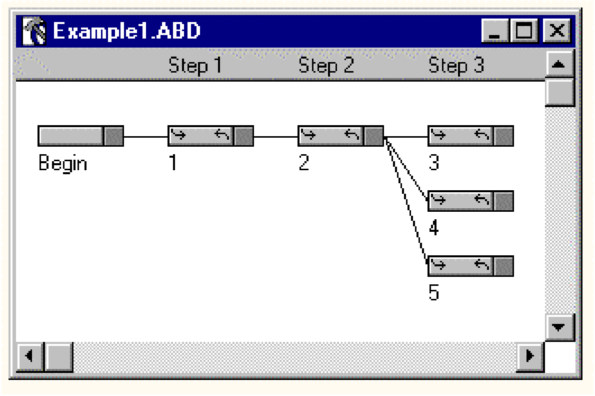


Figure 4-1 Example 1

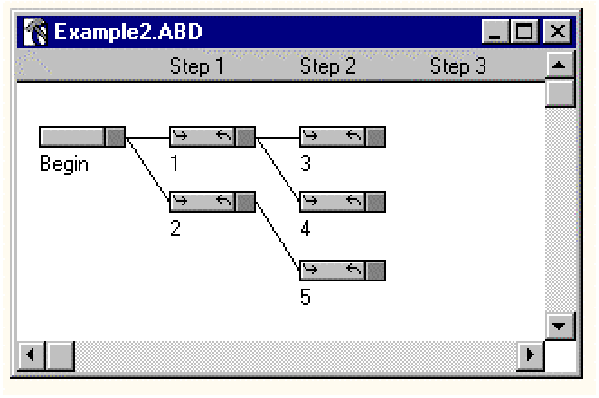


Figure 4-2 Example 2

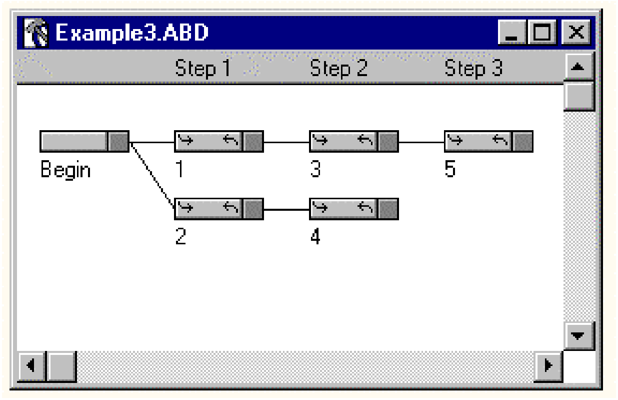


Figure 4-3 Example 3

Figure 4-4 shows a more complex application and the Input Handler's corresponding ID numbers.

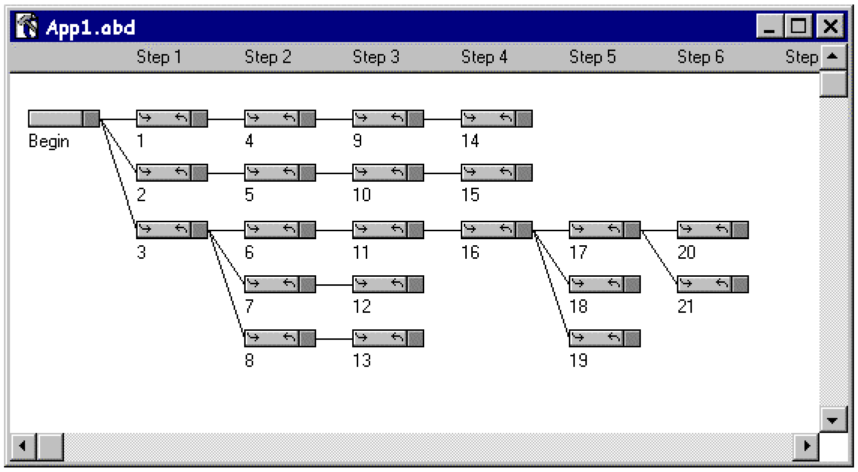


Figure 4-4 Input Handler IDs

Code Generation Directives

Each code generation directive is in the form:

directive = *name* [*symbol*] [“(“ argument { “,” argument }”)”]

name is the name of the directive (for example, **LASTSTEP**); ***symbol*** is only used in a few directives and identifies the name of a user-defined symbol. Symbols can be defined and then used as directives, where the value of the *symbol* is substituted for the *symbol* name. Following the name of the directive and optionally the *symbol* name, is an optional argument list. Arguments may be other directives.

Following is a comprehensive list of all the code generation directives that can appear in a Source Template file, with an explanation of each.

BSTRING

Takes a single text argument.
Converts text returned from some other directive into a BASIC-style string. For example, if the text of the first action of the first handler is:

```
line 1
line 2
```

then **`BSTRING(HACTTEXT(0, 0))`** will return the text:

```
"line 1" + chr$(13) + chr$(10) + "line 2"
```

CSTRING

Takes a single text argument.
Similar to **BSTRING**, but returns a C-style string. Using the example above, the result would be **`line 1\r\nline 2`**.

DEBUG

Takes a single numeric argument: a handler ID.
Notifies the source code generator that we're about to generate code for a handler with the given ID. The source code generator can then create a file with debugging information for the simulator. Any time one of the following lines of code is executed, the simulator asks Application Builder to select the numbered Input Handler. *(This directive is to support a future simulator in Application Builder. At the time of this release, Application Builder does not have a simulator.)*

EOLN

Takes no arguments.
Returns the text of the line-ending requested in the **Data Format** dialog box for the current application. Possible values are \r, \n, or \r\n.

EQ

Takes two arguments.
Returns 0 or 1 depending on whether the two arguments are equal.

ESCINPUT

Takes no arguments.
Returns non-zero if application should generate escape input on an exit event, rather than just quitting.

FNAME

Takes one argument.
Returns the name of the numbered file. Files are numbered from 0 through **LASTFILE**.

FOR
NEXT

FOR takes a counter name, two arguments, and an optional third argument. **NEXT** takes no arguments.
The symbol name after the **FOR** keyword is the name of the counter variable. The first two arguments to **FOR** are the first and last numbers for the loop counter. If a third argument is given, it's the number that should be added to the counter each time through the loop. For example, the Source Template:

```
~FOR COUNT(0, 3)~
line ~COUNT~
~NEXT~
```

generates the following source code:

```
line 0
line 1
line 2
line 3
```

FTYPE

Takes a single argument.
Returns the type of the numbered file. Files are numbered from 0..**LASTFILE**. Possible types are 0 for data file and 1 for CRF files.

HACTNUM

Takes two arguments: a handler ID and an action number.
Returns the text of the action, converted to a number.

HACTTEXT

Takes two arguments: a handler ID and an action number.
Returns the text of the action.

HACTTYPE

Takes two arguments: a handler ID and an action number.
Returns the type of action. The type can be 0 for good beep, 1 for bad beep, 2 for add data to record, 3 for add date to record, 4 for add time to record, 5 for add CRF field to record, 6 for add text to record, 7 for plug-in, 8 for quit, 9 for shut down, or 10 for set variable.

HAUTO

Takes a single argument: a handler ID.
Returns the index of the first handler following this one that has an automatic match criterion. Returns 0 if there is no such handler.

HDETOUR

Takes a single argument: a handler ID.
Returns true if the handler is a detour handler.

HDETOURTYPE

Takes a single argument: a handler ID.
Returns the type of detour handler it is, or -1. The type can be 0 for return after actions, 1 for return after timeout, or 2 for return after ESC.

HDEVICE

Takes a single argument: a handler ID.
Returns a handler's device matching property as a number. The number is any combination of the possible devices listed under **INPUTEVT** in the *Videx BASIC Manual*.

HELOOP

Takes a single argument: a handler ID.
Returns true if the handler ends a loop.

HFOLLOWID

Takes two arguments: a handler ID and a 0 based index.
Returns the index of the *n*th handler to which this handler is connected.

HLASTACT

Takes a single argument: a handler ID.
Returns the index of the last action in the handler’s list of actions. (If there are six actions it returns 5, since the actions are numbered from 0 through 5.)

HLASTFOLLOW

Takes a single argument: a handler ID.
Returns the index of the last handler to which this handler is connected. (If it’s connected to six handlers, it returns 5, since the handlers are indexed from 0 through 5.)

HLASTSFIELD

Takes a single argument: a handler ID.
Returns the number of fields in the handler’s display. (Actually returns the index of the last field, one less than the number of fields.)

HMATCHFILE

Takes a single argument: a handler ID.
Returns the file number for a handler’s CRF file matching, if the handler uses a CRF file for its match criterion.

HMATCHTEXT

Takes a single argument: a handler ID.
Returns the text of the match criterion.

HMATCHTYPE

Takes a single argument: a handler ID.
Returns the type of match criterion used by the handler. The type can be 1 for CRF file, 2 for automatic, 3 for starts with, 4 for equals, 5 for contains, 6 for number, or 7 for TWII pattern.

HNAME

Takes one argument: a handler ID.
Returns the name of the handler.

HPROMPTB
HPROMPTL
HPROMPTR
HPROMPTT

Takes one argument: a handler ID.
Returns the top, left, bottom, or right of the keyboard entry area in the display's screen.

HSCREEN

Takes three arguments: a handler ID, the number of rows, and the number of columns.
Returns the upper-left corner of the screen text of the handler (which is 2 rows by 16 columns), appending "\r\n" to the end of the first row.

HSFFIELD

Takes two arguments: a handler ID and a field index.
Returns the CRF field index of a display object.

HSFLEFT

Takes two arguments: a handler ID and a field index.
Returns the index of the leftmost character in the screen text where the display field should go, assuming that each line of the screen text ends with a carriage return and line feed.

HSFLEN

Takes two arguments: a handler ID and a field index.
Returns the width of the display field.

HSFSTEP

Takes two arguments: a handler ID and a field index.
Returns the step from which the display field should get its data.

HSFTEXT

Takes two arguments: a handler ID and a field index.
Returns the text of the display field.

HSFTYPE

Takes two arguments: a handler ID and a field index.
Returns the type of the display field. The type can be 6 for input, 7 for date, 8 for time, 9 for reference field, 10 for text, or 11 for plug-in.

HSLOOP

Takes one argument: a handler ID.
Returns true if the handler starts a loop.

HSTEP

Takes one argument: a handler ID.
Returns the handler's step.

HSYMBOL

Takes one argument: a handler ID.
Returns a handler's symbology matching property. The number is any combination of the possible symbology codes listed under **INPUTEVT** in the *Videx BASIC Manual*.

IF
ELSEIF
ELSE
ENDIF

IF and **ELSEIF** take one argument: a Boolean value. **ELSE** and **ENDIF** take no arguments.
Conditionally includes parts of the Source Template.

LASTFILE

Takes no arguments.
Returns the index of the last file used by the application. By convention, the data file is file number 0 and reference files are numbered from 1...**LASTFILE**.

LASTHANDLER

Takes no arguments.
Returns the index of the last handler in the application. Handlers are numbered from left to right and top to bottom, with the “Begin” Handler being number 0.

LASTSTEP

Takes no arguments.
Returns the index of the last step in the application. The “Begin” Handler is in step 0 and subsequent steps are numbered from 1..**LASTSTEP**.

NE

Takes two arguments.
Returns 0 or 1 depending on whether the two arguments are equal.

PROGRESS
PROGRESSSTART

PROGRESSSTART takes one argument: the number of stops.
PROGRESS takes no arguments.
PROGRESSSTART allows the code generation application to display a progress bar with n+1 locations on the bar. **PROGRESS** causes the code generation application to increment the progress bar to the next location.

QUOTES

Takes no arguments.
Returns a value indicating the usage for quotes in the data file. Possible values are 0 for none, 1 for around fields, or 2 for around records.

SEARCHDEPTH

Takes no arguments.
Returns the depth required for the deepest loop search. This basically returns the longest possible path through the application, returning the number of handlers in that path.

SEPARATOR

Takes no arguments.
Returns the character to be used as a field separator in the data file. Possible values are 0 for none, 1 for tab, 2 for comma, 3 for space, or 4 for end of line.

SET

Takes a symbol and one argument.
Creates a new symbol in the symbol table, and assigns it the value in the argument. The symbol can subsequently be used as a code generation directive, where the symbol name will be replaced by its value. For example, the Source Template:

```
~SET FOO(321)~
line 1
~FOO~
line 2
```

would generate the following source code:

```
line 1
321
line 2
```

The only other way to create symbols is using the **FOR** directive, which creates a temporary symbol for the duration of the **FOR** loop.

Describe.src

Describe.src is the template that Application Builder uses to generate BASIC source code. You can change the way Application Builder generates code by editing **Describe.src**. **Describe.src** should be kept in the same folder as **Appbuild.exe**.

The heart of **Describe.src** is the event loop, and the heart of the event loop is the call to **INPUTEVT**. One of the parameters returned by **INPUTEVT** is *evt_type%*. See the *Videx BASIC Manual* for more information on the **INPUTEVT** statement.

The **Describe.src** variables are described in the tables on pages 89–94. The **Describe.src** subroutines are described on pages 95–98 and the sounds used by **Describe.src** are described on page 99. The table on pages 100–101 is a list of commonly used plug-ins.

Describe.src Variables

Following is a list of variables used in **Describe.src**. (Note: This list does not include variables generated by code generation directives.)

Variable	Description
accepts%	Set to state number of accepting handler if input is accepted. Set to zero if input is not accepted. *(Remember to include the 's' in accepts%.)*
archive_data$()	Records data for each step in the design. Indexed by step.
archive_date$()	Records date for each step in the design. Indexed by step.
archive_time$()	Records time for each step in the design. Indexed by step.
archive_val1$()	Records first CRF field for each step in the design. Indexed by step.
archive_val2$()	Records second CRF field for each step in the design. Indexed by step.

Variable	Description
archive_val3$()	Records third CRF field for each step in the design. Indexed by step.
bad_dur%	Duration of “low” beep in milliseconds (ms). Set to 400.
bad_frq%	Frequency of “low” beep in Hertz (Hz). Set to 723.
bar$, bar2$	Temporary string variable used at various points in the program.
cstate%	Current state.
cstep%	Current step.
display$	Current string to display on the screen.
display_flag%	Counter which cycles from 0 to 5 and determines what piece of system information to display when the scroll down key is pressed.
eoln$	Delimiter used at end-of-line in data file. Set to <cr><lf>.
evt_data$	Input data most recently returned by **INPUTEVT**.
evt_device%	Device most recently returned by **INPUTEVT**. Returns 1 for keyboard, 2 for laser, 12 for contact scanner, 48 for touch button, 63 for automatically generated input, or 64 for application generated input (mock <<Escape>> input).
evt_symbol%	Bar code symbology of input most recently returned by **INPUTEVT**. Returns 1 for USS-39, 2 for UPC-A or UPC-E, 4 for USS-I2/5, 8 for Codabar, 16 for EAN, 64 for UCC Code 128, 128 for USS-128, 159 for mock input event (software generated <<Escape>> input or automatically generated input), or -1 if <f1> or <f2> key is pressed on the LaserLite Pro or LaserLite Mx.

Variable	Description
evt_type%	Type of event most recently returned by **INPUTEVT**. Returns 1 for input, 2 for exit, 3 for scanning a five space bar code that deletes last input, 4 for scroll up, 5 for scroll down, 6 for power off, 7 for unexpected power loss detected, 8 for ESC key, 9 for scan key pressed and released, 10 for LaserLite Mx no memory card module found, 11 for LaserLite Mx no memory card found, 12 for LaserLite Mx unknown memory card found, 13 for LaserLite Mx memory card ID has changed, 14 for LaserLite Mx memory card processor interrupted, 18 for scroll left, 19 for scroll right, 20 for MEM key, 21 for BAT key, 22 for f1 key, 23 for f2 key, 24 for f3 key, 25 for f4 key, or 26 for f5 key.
field_del$	Field delimiter returned by Application Builder. Can be none, tab, comma, space, or line-ending returned by Application Builder (\r, \n, or \r\n).
field_quote$	Contains quotes if fields are to be quoted; otherwise empty string.
foo%	Temporary integer variable used at various points in the program.
full_display$	Contains complete string to be displayed. Used by the scroll left and scroll right routines.
good_dur%	Duration of "good" beep in ms. Set to 250.
good_frq%	Frequency of "good" beep in Hz. Set to 1446.
high%	Total size (in bytes) of data file in multiples of 32,768 (high bytes). Used in subroutine *truncate* when removing bytes from the data file.
key_dur%	Duration of click in ms when entering keypad data. Set to 10.
key_frq%	Frequency of click when entering keypad data. Set to 1397.

Variable	Description
loop_index%	Index of the state in *path%()* that immediately precedes the accepting state. Generated by subroutine *loop_search*, which searches for a handler to accept the current input.
loop_state%	Temporary integer variable used in subroutine *loop_search* when testing whether states in *path%()* will accept the current input.
low%	Total size (in bytes) of data file modulo 32768 (low bytes). Used in subroutine *truncate* when removing bytes from the data file.
low_voltage%	Voltage at which unit gives warning message. Set at 45 for 4.5 volts; do not use less than 35.
max_code%	Maximum length of user input. Maximum is 64 characters; greater values are not recognized by the operating system.
n_times%	Used in *kill_time* subroutine together with *x_delay%* to determine length of delay when displaying messages to user.
nbytes%	Number of bytes in current record under construction.
old_state%	State immediately preceding current state. Used when deleting data from the data file.
path%()	Stack of states containing handlers that have been visited. (Each handler has a unique state number. The "Begin" Handler is state 0 and step 0. In general, if there is any branching in an application, the number of states is greater than the number of steps.)
path_top%	Index of *path%* corresponding to top of stack. Initialized to -1 (pointing below the bottom of the stack). The "Begin" Handler corresponds to *path_top%=0*.

Variable	Description
pop_count%	Number of states to pop off the stack.
prompt_b%	Defines the bottom of the screen rectangle where data entry occurs. Used by **INPUTEVT**.
prompt_l%	Defines the left side of the screen rectangle where data entry occurs. Used by **INPUTEVT**.
prompt_r%	Defines the right side of the screen rectangle where data entry occurs. Used by **INPUTEVT**.
prompt_t%	Defines the top of the screen rectangle where data entry occurs. Used by **INPUTEVT**.
push_state%	State number of the handler to be pushed onto the *path%* stack. Initialized to zero when pushing the "Begin" Handler onto the stack.
record$	Current record. Maximum length of a record is 128 characters.
record_end$	Contains string to be written at the end of each record in the data file.
record_len%()	Length of the record at each state on the stack. Used to help truncate previous record when user doesn't loop all the way back to the first step.
record_quote$	Contains quotes if records are to be quoted; otherwise empty string.
ref_error_name$	Name of the reference file that results in an error condition when opening.
return_top%	Records state number at top of stack just before pushing a detour. Set to -1 when not stepping to a detour handler.
running%	Set to *true%* (1) while running application. Set to *false%* (0) when exiting application (**Unlock** command received by **INPUTEVT**).
scroll_mode%	Current scroll status. Can be *scrolling_none%, scrolling_up%, scrolling_down%,* or *scrolling_limit%.*

Variable	Description
scrolling_down%	Setting for scrolling down through the data file. Set to 3.
scrolling_limit%	Setting for having scrolled to the top of the data file. Set to 4.
scrolling_none%	Setting for not scrolling. Set to 0.
scrolling_up%	Setting for scrolling up through the data file. Set to 2.
shift%	Used in scroll left and scroll right routines to determine which portion of *full_display$* is to appear in the display.
state_auto%()	Indexed by state. Contains true if that state has an automatic matching handler following it.
state_eloop%()	Indexed by state. Contains true if state ends loop.
state_step%()	Indexed by state. Contains the state's step.
token_del$	Quote + field_del$ + line-ending returned by Application Builder (\r, \n, or \r\n).
voltage%	Voltage (saved from the last time the voltage was read). The user gets a low-voltage warning whenever the voltage changes and is below *low_voltage%* (4.5 volts). Threshold voltage is 3.5 volts.
x_delay%	Used in *kill_time* subroutine together with *n_times%* to determine length of delay when displaying messages to user.

Describe.src Subroutines

Following is a list of subroutines used in **Describe.src**.

Subroutine	Purpose
accept*x*	Determines whether state *x* accepts current input. This routine is automatically generated for each state.
action*x*	Performs actions for state *x*. This routine is automatically generated for each state.
bad_tone	Sounds two tones indicating the data is invalid.
check_voltage	Checks the battery voltage. If it is less than 4.5 volts and if it has changed since the last time it was checked, warns the user.
d_find*x*	Attempts to find a detour handler to step to from state *x*. This routine is automatically generated for each state.
delete_record	Deletes the last record in the data file after warning the user.
deletetone	Sounds two tones indicating that the data was deleted from the data file.
dispatch_display	Given the current value of *state%,* dispatches to the proper display routine.
display_info	When scrolling down and at the end of the file, cyclically displays battery voltage, time, date, OS version, ID, and available memory.
display*x*	Prepares display for state *x*. This routine is automatically generated for each state.
evt_bat	Responds to a battery event (generated by pressing the BAT key on the LaserLite Mx or LaserLite Pro) by clicking and displaying the battery voltage.

Subroutine	Purpose
evt_delete	Responds to a delete event or to pressing the f5 key on the LaserLite Mx or LaserLite Pro, by deleting the most recently entered data (after warning the user). If already at the beginning of the record, delete the previous record of the data file (after warning the user).
evt_down	Responds to the scroll-down key by clicking and then scrolling down, first through the data file, then the current record, and then cyclically displaying the battery voltage, time, date, ID, OS version, and remaining memory.
evt_esc	Responds to an escape event, generated by pressing the ESC key on the LaserLite Mx or LaserLite Pro. Clicks, and then, if in a detour handler, returns to a normal handler.
evt_f1	Responds to a press of the f1 key by simulating a user input of <<F1>>.
evt_f2	Responds to a press of the f2 key by simulating a user input of <<F2>>.
evt_f3	Responds to a press of the f3 key by simulating a user input of <<F3>>.
evt_f4	Responds to a press of the f4 key by displaying the version of the operating system in the unit.
evt_input	Responds to an input event by searching for a state to accept the current input.
evt_left	Responds to the scroll-left key by clicking and then scrolling to the left to display a long entry.
evt_mem	Responds to a memory event (generated by pressing the MEM key on the LaserLite Mx or the LaserLite Pro) by clicking and displaying free memory.
evt_none	Does nothing. Branched to from **INPUTEVT** if an event type occurs that requires no action to be performed (event types 0, 9–14).

Subroutine	Purpose
evt_pfail	Responds to an unexpected loss of power (power lost when unit was not asleep). Displays a warning message, makes a ringing sound, and puts unit to sleep.
evt_right	Responds to the scroll-right key by clicking and then scrolling to the right to display a long entry.
evt_switch	Responds to a flip of the lock switch to the OFF position by turning off the display and putting the unit to sleep.
evt_unlock	Responds to an **Unlock** command by either quitting the application or by generating a mock <<ESCAPE>> input.
evt_up	Responds to the scroll-up key by clicking and then scrolling up, first through the current record, then the data file.
file_error	Displays a message alerting user to error accessing the data file, rings, and puts the unit to sleep.
finish*x*	Performs housekeeping when stepping to state *x*. This routine is automatically generated for each state.
kill_time	Delays program operation while information is being displayed.
l_find*x*	Attempts to find a handler to loop back to from state *x*. This routine is automatically generated for each state.
ld_find*x*	Returns from state *x* (which is a detour handler) to previous state in path stack. This routine is automatically generated for each state.
loop_search	Searches the *path%()* stack for a handler which accepts the current input.
loop_to	Loops the application back to the accepting handler found by *loop_search*.

Subroutine	**Purpose**
name*x*	Sets display to show name of state *x*. This routine is automatically generated for each state.
pop	Pops *pop_count%* states off the stack. Truncates the current record and sets the display.
push	Pushes a state onto the *path%()* stack. The current input is stored in the archives, its actions are performed, and assignments are made to *display$.*
questiontone	Sounds two tones when asking user to confirm that data is to be deleted from the data file.
ref_error	Displays a message alerting user to error opening a CRF file, rings, and puts unit to sleep.
return_to	Returns from a detour handler to a normal handler.
ring	Makes a ringing sound to alert the user to an error.
s_delete*x*	Deletes the information written to the data file by state *x*.
s_find*x*	Attempts to find a handler to step forward to from state *x*. This routine is automatically generated for each state.
truncate	Removes *nbytes%* bytes from data file.

Describe.src Sounds

Following is a list of sounds used in **Describe.src**.

Sound	Code	Events
Three beeps: high, low, high	sound 2793, 250 sound 2637, 250 sound 2793, 250	Begin application (startup beep).
Three beeps: high, medium, low	sound 1760, 250 sound 1568, 250 sound 1397, 250	Quit application (exit beep).
Two beeps: medium, high	sound 2349, 150 sound 3620, 250	Ask user to confirm that data is to be deleted from the data file.
Two beeps: high, medium	sound 3620, 150 sound 2349, 250	Data is being deleted from the data file.
Two beeps: low, very low	sound 723, 500 sound 578, 600	No state will accept the input data (invalid data).
Single high beep	sound 2349, 250	1. User cancels attempt to delete data. 2. User attempted to delete data, but data file was empty.
Single medium beep	sound 1446, 250	Input accepted by a handler (and user requested a “beep” action).
Single low beep	sound 723, 400	Input accepted by a handler (and user requested a “low beep” action).
Single click	sound 1397, 10	A key was pressed.
Ring	for ri%=1 to 10 sound 2093, 50 sound 2794, 50 next ri%	1. Power lost when unit was not asleep. 2. Low-battery voltage detected. 3. Error accessing the data file or a CRF file.

Plug-ins

Plug-ins are BASIC source code that is inserted into the application by Application Builder. Plug-ins provide additional features when creating applications, but they do require familiarity with BASIC programming and the **Describe.src** source code template.

Plug-in	**Function**
if *condition* then accepts% = false%	If *condition* is true, the current input is rejected.
if *condition* then accepts% = true%	If *condition* is true, the current input is accepted.
evt_data$ = *expression*	Set input data to *expression*. (Requires that the Input Device property include **Software**.)
gosub evt_delete	Delete last scan.
gosub ring	Unit makes a ringing sound.
volt1% = *variable%*	Change battery voltage warning message trigger to *variable%*. Default is 48, corresponding to 4.8 volts.
print #0, *expression*	Write *expression* to data file.
running% = false%	Quit application.
sound *frequency%*, *duration%*	Unit generates a sound at *frequency%* Hz for a duration of *duration%* milliseconds. The *frequency%* must be between 50–8000 and *duration%* must be between 1–2000.

Plug-in	**Function**
*variable$* = environ$(0)	Place current battery voltage in *variable$.*
*variable$* = environ$(1)	Place available RAM in *variable$.*
*variable$* = environ$(2)	Place operating system version in *variable$.*
*variable$* = environ$(3)	Place unit's ID in *variable$.*

Timewand.src

Timewand.src is a version of the template similar to **Describe.src**, except **Timewand.src** generates a TimeWand II-style data file. As a result, it handles deletion and scrolling somewhat differently.

To generate a TimeWand I-style data file instead of a TimeWand II file, remove all of the lines between "TWII BEGIN" and "TWII END" in the **Timewand.src** source file before renaming it. (Note: You will lose the ability to delete scans.)

Notes:

Chapter 5

LaserLite Mx Memory Card Data Management

This chapter contains information on:

- The capabilities of the memory card.
- LaserLite Mx memory card command set.
- Transferring files between the memory card and the computer.
- Booting from the memory card.
- Sending cross-reference files to the memory card.
- Transferring data from the memory card.
- Creating and using cross-reference files.
- Opening a memory card file.
- Changing records in a memory card file.
- Handling errors.

Memory Card Capabilities

You can perform the following operations with a LaserLite Mx and a memory card:

- Create, open, close, and delete files. Four different types of files are supported: *identification* (**D**), *boot* (**B**), *sequential* (**S**), and *indexed* (**I/H**). *(See the table on page 136 for descriptions of the different file types.)*
- Add, delete, and change records.
- Search records based on the key field.
- Move pointer within a file.
- Seek information based on a given string.
- Reorganize space.
- List memory card file information and status report.
- Boot main CPU

You can use either the **Vxcom** *(Windows 95/98/NT)* or **Download** *(DOS)* communications program's commands file to transfer files to and from the memory card.

LaserLite Mx Memory Card Command Set

Operating System P Command–Pass to Memory Card

The **P** command is the **Pass to Memory Card** command for the LaserLite Mx operating system software command line. The **P** command allows the communications program to directly control interaction with the memory card file system.

Syntax:

P <*command string*>

The **P** command sends the accompanying command string to the memory card and returns the response as received from the memory card. The operating system can generate a special error, -2, to indicate that a memory card was not found. Otherwise, it returns the response exactly as received from the memory card for each command sent.

Both the commands file **R** command (described on pages 106 and 133) and **S** command (described on pages 106 and 134) use a series of **P** commands to communicate with the memory card. On the following page are descriptions that show how the **P** command interprets the **R** and **S** commands.

R Command (Commands File) Actions:

1. The communications program sends a file to the memory card by issuing a series of **P** commands:
   - It begins with the open (**O**) command:

     `P o` *`<modulus> <file type> <filename>`*
   - It then writes records using the append (**A**) command:

     `P a` *`<data strings>`*
2. The communications program processes and reports any errors generated by the memory card processor or the operating system.

S Command (Commands File) Actions:

1. The communications program retrieves a file from the memory card by issuing the following series of **P** commands:
   - It opens the file with the open (**O**) command:

     `P o` *`<modulus> <file type> <filename>`*
   - Then the communications program moves the pointer to the top of the file with the move pointer (**M**) command and retrieves the first record:

     `P m &FFFF R`
   - It then continues to issue **M** commands:

     `P m 1 F` *or* **`P m 1024 F`**

     until it reaches the end of the file.
2. The communications program processes and reports any errors generated by the memory card processor or the operating system.

The **P** command uses the same command strings that are used by the Videx BASIC **CARDCMD** statement's *command_str$* parameter. The following table gives a brief description of these commands along with the page number where you can find complete information about the command. Parameters within { } are required; parameters within [ ] are optional.

Command String	**Description**
A *[hash value] {record}*	Add a new record to the memory card's open file. See page 109 for more information.
C *{F/S/I} [status bytes]*	List or send the card's file management report or status information. See pages 110–113 for more information.

D *{file type} {filename}*
D *{file handle}*
D *{file handle} [file type] [filename]*
	Delete a file from the memory card.
	Delete an existing file.
	Rename an existing file.
	See page 114 for more information.

F *[hash value] [key field]* **F**	Search for a record in the memory card's open file with the given key field and send it to the host. Search for next record with same hash value or key field. See page 115 for more information.
H *[hash value] [key field]* **H**	Delete a record from the memory card's open file with the given key field. Delete next record with same hash value and key field. See page 116 for more information.
K &1092	Remove all deleted files from the memory card. See page 117 for more information.

M *{# of records/bytes} {F/R}*
M
M H *{file handle}*
	Move the pointer within the open file.
	Show the current record.
	Delete the record at the move pointer.
	See pages 118–120 for more information.

command_str$	**Brief Description**
N *[param1]*	Format new memory card or determine the memory card ID. See page 121 for more information.

O *[modulus] {file type} {filename}*
 	
O *{file handle}*
	Open a new or existing file on the memory card.
	Open an existing file. See pages 122–123 for more information.

Q *{file type} {filename}*	Calculate the CRC of a file and send it back to the host. See page 124 for more information.
S *{field #/bytes} {F/R} {string}*	Perform search within the card's open file. See page 125 for more information.
V	Read the memory card's program version. See page 126 for more information.
Y	Repeat the last status byte or data. See page 127 for more information.
Z	Puts the data collector to sleep. See page 127 for more information.

The commands are described in detail in the following sections.

A *[hash value] {record}*

Adds data to the memory card's open file (see the **O** command on pages 122–123 to open a file). For an indexed (**I**) file, the card module first calculates the hash entry based on the key field of the *record* and then looks up the hash table for a pointer. If no pointer is found, it is the first record for the entry. The *record* is saved as the current end-of-file and the pointer at the hash table is set. If a pointer is found, a collision occurred for this entry. The DMS traverses the list and finds the last record with the same *hash value*. The new record is saved as the current end-of-file and the pointer is set at the last record.

For type **I** files, the *hash value* is calculated internally. You can have more than one record with the same key field. The key field is delimited from other fields with a tab character (09 hex). For type **H** files, the *hash value* must also be passed with the command. For type **S**/**D**/**B** files, the **A** command appends *{record}* as data.

Example: Add a record to the open indexed file **test1**:

```
P a "This is a record"
```

Returns:

00	Record added.

Successful return values:

00

Possible error codes that can be returned from this command:

01, 03, 04, 10, 32, 34, 39, 40, 41, 42, 53, 61, 63 *(See pages 128–129 for a description of the error codes.)*

C *{F/S/I} [status bytes]*

Lists the memory card's file management or status information or sends a setup configuration to the memory card processor.

> ***F*** - List the file management report.
> ***S*** - List the status report.
> ***I*** - Send the report to the memory card processor to update.

F - List the file management report

The file management report is sent out first, if asked, followed by individual file information. The filenames are enclosed in quotation marks. The file type is sent out as a hexadecimal value, preceded by an ampersand. All hexadecimal values returned by the memory card begin with an ampersand (&) character. A space character is used to separate hexadecimal from ASCII, or different parts of the listing (for example, a space between file 1 and file 2). The following tables show the order of the data format. A sample of the file management report is shown on the following page.

File Management Report Format:

bytes 1–2	Total space of the card (Kbytes).
bytes 3–4	Bad space (Kbytes).
bytes 5–6	Available space (Kbytes).
bytes 7	Number of files (including deleted files).
bytes 8–b1	1st valid file information.
bytes b2–b3	2nd valid file information.
bytes b4–b5	3rd valid file information.
...	

Individual File Information:

byte 1	File serial number in the memory card.
byte 2	File status.
byte 3	File type.
bytes 4–5	File table size.
bytes 6–7	The file size (Kbytes).
bytes 8–16	The filename (variable length from 1–18 bytes).

Example: Request file management report:

```
P c f
```

Returns:

```
&07980000076805 &01FF44FFFF0004 "card1"
&02FF42FFFF0014 "CRD" &030049089B000C
"data.txt" &03FF49089B000C "olddata.txt"
&04FF49089B000C "data.txt"
```

Note: The **F** command in a commands file issues the **C F** command to the memory card. It creates the following report on the information above:

```
Memory Card File and Memory Status Report

Total Space:         1944 Kbytes
Bad Space:              0 Kbytes
Available Space:     1896 Kbytes
Retrievable Space:     12 Kbytes

Number of deleted files:   1
Number of valid files:     4

File Number  Deleted  Type  Table Size  Kbytes  Name
     1         No      D         n/a        4   "card1"
     2         No      B         n/a       20   "CRD"
     3         Yes     I        2203       12   "data.txt"
     3         No      I        2203       12   "olddata.txt"
     4         No      I        2203       12   "data.txt"
```

S - List the status report

At any given point in time, the LaserLite Mx can have an open file and a pointer to data or to a record within that file. When power is removed from the memory card module during sleep, the data management system loses track of this information. The status report contains the information needed to restore the memory card system to a previous state.

Status Report Format:

byte 1	Last command before the **C** command.
bytes 2–6	The move pointer: block # (2 bytes), page number (1 byte), column address (2 bytes).
byte 7	File type.
bytes 8–b1	Opened filename (variable length).

Example: Request status report:

```
P c s
```

Returns:

```
&4F001503006A04
```

I - Send the status report to the memory card processor to update

The status string retrieved by the **C S** command may be sent to the memory card processor with the **C I** command. The LaserLite Mx operating system and BASIC language automatically update the state of the memory card in its sleep/wake routines. Normally, you will not be concerned with updating status, except when developing complex operations using multiple files.

Example: Restore the status after waking from sleep:

```
P c I &4F001503006A04
```

Returns:

`00`	Previous status restored.

Successful return values:

File management report (for **C F**); status report (for **C S**); 00 (for **C I**).

Possible error codes that can be returned from this command:

01, 03, 04, 10, 31, 51, 52, 63 *(See pages 128–129 for a description of the error codes.)*

D *[modulus] {file type} {filename}*

or

D *{file handle}*

or

D *{file handle} {file type} {new filename}*

Deletes or renames a file on the memory card. To delete a file, list the *file type* (indexed (**I** or **H**), sequential (**S**), identification (**D**), or boot (**B**)) and *filename* of the file to delete. The *filename* can have up to 18 bytes, or you can use the *file handle* number (if known from a previous **O** command).

To rename a file, you must refer to the file by the *file handle* number and *file type*, and pass it the new *filename*.

Example 1: Delete an indexed file with *filename* **data.txt:**

```
P d I "data.txt"
```

Returns:

00	File deleted successfully.

Example 2: Rename an indexed file with *filename* **data.txt** and *file handle* **03** to **olddata.txt:**

```
P d 03 I "olddata.txt"
```

Returns:

00	File renamed successfully.

Successful return values:

00

Possible error codes that can be returned from this command:

01, 03, 04, 09, 10, 31, 33, 40, 63, 65 *(See pages 128–129 for a description of the error codes.)*

F *[hash value] [key field]*

or

F

Searches for a record with the given *key field* in the memory card's open file and sends it to the host. This command only functions with file types **I** and **H**. When a *key field* is not provided, and the last command was an **H** or an **F** command with a *key field*, it tries to read the next record with the same *key field* as was used in the last **H** command. For example, to read multiple records with the same *key field*, the following commands can be used:

Assuming there are three records with the *key field* "ABCDEFG" in an indexed file, then:

`P f "ABCDEFG"`: reads the first record;

`P f`: reads the second record;

`P h`: deletes the third record.

To read all of the records in an indexed file with the same *hash value*, the following commands can be used:

`P f` *[key field]***:** first record in the chain;

`P f`: second record in the chain.

Example: Find a record in the open indexed file **test1:**

```
P f "abc"
```

Returns:

`"abc" <field2> <field3>`	Record found.
`35`	No records match the find request.

Successful return values:

A record.

Possible error codes that can be returned from this command:

01, 03, 04, 06, 08, 10, 32, 35, 53, 63 *(See pages 128–129 for a description of the error codes.)*

H *[hash value] [key field]*

or

H

Deletes a record with the given *key field* from the memory card's open file. This command only functions with file types **I** and **H**. For an indexed file, the program goes through the hash table, finds the record, and sets the record status bit to 0. It does not change the pointer or erase the record. When a *key field* is not provided, and the last command is an **F** or an **H** command with a *key field*, it tries to delete the next record with the same *key field* as in the last **H** command.

Example: Delete a record from the open indexed file **test1:**

```
P h "abc"
```

Returns:

`0`	Record deleted.

Successful return values:

00

Possible error codes that can be returned from this command:

01, 03, 04, 06, 08, 10, 32, 35, 39, 40, 53, 63, 65 *(See pages 128–129 for a description of the error codes.)*

K &1092

Removes deleted files from the memory card. This command copies the file management report from the control blocks to other blocks, changes content, erases the control blocks, and copies the information back to the control blocks. Do not interrupt the processor (power down or reset) during the operation or you could lose file management information. This command should only be used after important data has been transferred from the memory card to the computer.

Example: Clean up the card:

```
P k &1092
```

Returns:

`00`	Deleted files removed from the memory card.

Successful return values:

00

Possible error codes that can be returned from this command:

01, 03, 04, 10, 40, 61, 62, 63, 65, 67, 68 *(See pages 128–129 for a description of the error codes.)*

`M` *{number of records/bytes} {F/R}*

or

`M`

or

`M H` *{file handle}*

Moves the pointer within the memory card's open file. For indexed (**I** or **H**) files, this command moves the pointer forward (F) or backward (R) a certain *number of records* within the open file, and sends out the current record that the pointer is pointing to. When *number of records* = **FFFF**, the pointer is moved to the end-of-file (F) or the beginning-of-file (R). For sequential (**S**) and identification (**D**) files, the number is in bytes, and the program sends out 256 bytes. When the *number of bytes* **= 0** (or is omitted), the program sends out the current record or 256 bytes, but does not move the pointer.

When a file is opened, the move pointer is set at the end-of-file. An **F** or an **H** command sets the move pointer to that record.

Example 1: Move forward 128K records from current position in open indexed file **test1:**

```
P m &020000 F
```

Returns:

`"abcdefg" <field 2> <field 3> <…>`	Record found; data reported.
`36`	End-of-file encountered.

Example 2: Move to beginning of the open indexed file **test1**:

```
P m &FFFF R
```

Returns:

`"abcdefg data"`	Record found, data reported.
23	There is no data in the file.

Example 3: Delete the record at the move pointer:

```
P m h
```

Returns:

`00`	Record deleted.

Successful return values:

A record or 256 bytes of binary data.

Possible error codes that can be returned from this command:

01, 03, 04, 10, 23, 36, 37, 39, 40, 53, 63, 65 *(See pages 128–129 for a description of the error codes.)*

The Move Pointer

The **M** card command moves a pointer within an open data file. The following table indicates the location of the move pointer under different circumstances.

Condition	**Location for file types I/H (indexed files)**	**Location for file types B/D/S (sequential files)**
Open file	Last record.	Last 256 bytes.
Append record	Last record.	Last 256 bytes.
Scrolling	Current record.	Current 256 bytes.
Closed file	Invalid.	Invalid.
Open file with no data	"No data condition" (error 23).	"No data condition" (error 23).
Delete record with the **H** command or **M H** command	At the deleted record. Must issue another **M** command before issuing an **M H** command. The **M** command then moves to the next record in the data file. If there is not a valid next record, then the move pointer moves to the previous record.	Not applicable.
F command	At the found record, or if not found, at the bottom of the data file.	Not applicable.
S command	At the found record, or if not found, at the bottom of the data file.	Not applicable.

N *[param1]*

Formats the memory card. **N**, by itself, closes any open files and restarts the memory card program. It also locks some of the physical functions, so the IRAM, XRAM, and memory card contents cannot be changed accidentally by using low-level routines. Issuing the **N** command without a parameter, returns the ID of the memory card.

The parameter **&0129** unlocks low-level physical functions permitting writing directly to the card module's XRAM memory and formatting the memory card.

The parameter **&1092** erases the entire card and formats the card, if it is not formatted.

Example 1: Format memory card:

```
P n &1092
```

Returns:

`0`	Memory card formatted.

Example 2: Determine memory card ID:

```
P n
```

Returns:

`"092612456"`	The card ID name, card module formatted.

Successful return values:

ID of the memory card.

Possible error codes that can be returned from this command:

01, 03, 04, 07, 10, 22, 63, 65 *(See pages 128–129 for a description of the error codes.)*

O *[modulus] {file type} {filename}*

or

O *{file handle}*

Opens an existing or a new file on the memory card. To open an existing file, only the *file type* (indexed (**I/H**), sequential (**S**), identification (**D**), or boot (**B**)) and the *filename* is needed, or you can use the *file handle* number (if known from a previous **O** command). The *modulus* (number of hash table entries) can be zero. The *modulus* can be omitted for sequential, identification, or boot files; for indexed or hashed files the *modulus* can be from 1–65535. (Note: See pages 141–142 for information on the modulus.) The *filename* can have up to 18 bytes. When a file is opened, any following operations are directed to the open file; any previously opened files are closed. When the module is powered up, the firmware boots the memory card and starts to run the DMS program from the XRAM. If successful, the **O** command returns the *file handle* number of the opened file.

One memory card ID file can be opened per card. The *filename* is used as the card ID, and any contents can be written to the file as a sequential file. The ID can be found by writing an ID file to the memory card with any name. It can also be found by issuing an **N** command (see page 121) without a parameter.

Example 1: Open an indexed file with table size 2203 and filename **test1:**

```
P o 2203 I "test1"
```

or

```
P o &089B I "test1"
```

or

```
P o 03
```

Returns:

`&03`	File opened successfully; **03** hexadecimal is file handle.
`39`	Memory full.

Example 2: Open the ID file with the filename **Videx**:

```
P o D "Videx"
```

Returns:

`&01`	ID file created successfully, with **Videx** as the card ID and assigned to file handle 01.
`"092612456"`	An ID file could not be created because an ID file already exists with **092612456** as the real card ID.

Successful return values:

File handle number or card ID name.

Possible error codes that can be returned from this command:

01, 03, 04, 07, 10, 31, 33, 39, 40, 51, 52, 63 *(See pages 128–129 for a description of the error codes.)*

Q *{file type} {filename}*

Calculates the CRC of a file and send it back to the host. This command is useful when sending a file to the memory card from the computer. **Vxcom** automatically calls this command when sending files to the memory card.

Example: Check the CRC of a boot file:

```
P q B "CPU"
```

Returns:

&B001	The 16-bit CRC of the boot file (boot files on the LaserLite Mx system must always return a CRC of **&B001**).

Successful return values:

16 bit CRC of the file.

Possible error codes that can be returned from this command:

01, 03, 04, 10, 31, 32, 51, 52, 63 *(See pages 128–129 for a description of the error codes.)*

S *{field number/bytes} {F/R} {string}*

Performs a search within the memory card's open file. For an indexed file (type **I** or **H**), this command searches for a record within the given number of records with the given string. The *field number* limits the search at the particular field. A zero *field number* forces the program to search through the entire record for the given string. The key field is the number one field. The *field number* is the fourth byte of the first parameter and the number of records that were searched is given as the three least significant bytes of the first parameter.

For a binary file, this command searches through the *number of bytes* trying to match the given *string*. Without any parameters, it reads the last record or 256 bytes found. This command uses the same pointer as the **M** command. Since this command does not use indexing, there may be some noticeable delay when using this command to search through large amounts of data.

Example: Search the open indexed file **test1** for a record in field two with the pattern "abcdefg" within the next 129838 records from the current move pointer:

```
P s &0201FB2E F "record"
```

Returns:

`"abcdefghijklmnop"`	Record found.

Successful return values:

A record or 256 bytes of binary data containing the search string.

Possible error codes that can be returned from this command:

01, 03, 04, 10, 23, 35, 36, 37, 39, 40, 63, 65 *(See pages 128–129 for a description of the error codes.)*

V

Reads the memory card's program version. The version is sent out in ASCII format. The version number for the DMS is:

```
VTDMSx.xx
```

The version number of the firmware is:

```
VTMCFx.xx
```

Example: Read the DMS version:

```
P v
```

Returns:

"VTDMS1.02"	The DMS version.

Successful return values:

Version number of the DMS or firmware.

Possible error codes that can be returned from this command:

03, 04, 10 *(See pages 128–129 for a description of the error codes.)*

Y

Repeats the data sent by the memory card processor.

Example: If the last command was the **V** example described above:

```
P y
```

Returns:

`VTDMS1.02`	The DMS version.

Successful return values:

Last data.

Possible error codes that can be returned from this command:

03, 04, 10 *(See pages 128–129 for a description of the error codes.)*

Z

Puts the data collector in sleep mode.

Example: Put the data collector to sleep:

```
P z
```

Returns:

`00`	Data collector put to sleep.

Successful return values:

00

Possible error codes that can be returned from this command:

03, 04, 10 *(See pages 128–129 for a description of the error codes.)*

The following table lists the status and error codes:

Code (Mem Card)	**Code (Comm SW)**	**Description**
00	21000	Operation successful.
01	21001	Unrecognized memory card. This version recognizes Toshiba's SSFDC 2, 4, and 8 MB memory cards.
02	21002	Unrecognized memory card.
03	21003	Syntax error.
04	21004	CRC of command did not match.
05	21005	Unknown command.
06	21006	Missing parameters for this command.
07	21007	Incorrect parameters for this command.
08	21008	Binary file.
09	21009	Incorrect file type.
10	21010	Too much data in the command.
22	21022	No ID file.
23	21023	No data.
31	21031	No such file exists.
32	21032	No file opened.
33	21033	Too many files (>60 files).
34	21034	The page has already been written to four times. (Note: Toshiba only allows you to write to a page four times.)
35	21035	No such record exists.
36	21036	End of file.
37	21037	Beginning of file.
38	21038	Timeout occurred.
39	21039	Memory full.
40	21040	Write/protect encountered.
41	21041	Record too large (>1024 bytes).
42	21042	Field too large (>255 bytes).
43	21043	Some of the physical functions are locked to avoid data corruption. Try: **N &0129** to unlock.

Code (Mem Card)	**Code (Comm SW)**	**Description**
51	21051	File management error (control data was changed).
52	21052	Sequential file corrupted (by accidental power failure).
53	21053	Data corrupted.
54	21054	CRC of boot file did not match (program data may be corrupted).
55	21055	The boot file program is too large (must be less than 24 KB).
61	21061	Memory card program failed (card may be worn out).
62	21062	Block erase failed (card may be worn out).
63	21063	Unknown memory card format.
64, 65	21064, 21065	First block of memory is bad (broken card).
66	21066	Too many bad blocks (card may be damaged).
67	21067	Most of the reserved control blocks are bad (card worn out).
68	21068	The memory card has been erased more than 32,768 times (each **K &1092** or **N &1092** counts as one erasing).

Transferring Files Between the Memory Card and the Computer

To transfer files to and from the memory card, you must use either the **Vxcom** *(Windows 95/98/NT)* or **Download** *(DOS)* communications program's commands file. **Vxcom** and **Download** are discussed in Chapter 1.

The syntax for the **Vxcom** command line is:

Vxcom*xxx* [app.s] [REF.CRF] [sys.os] [cmd.txt] {arguments}

or

Vxcom*xxx* [image.img] [cmd.txt] {arguments}

The syntax for the **Download.exe** command line is:

DOWNLOAD [app.s] [REF.CRF] [sys.os] [cmd.txt] {arguments}

or

DOWNLOAD [image.img] [cmd.txt] {arguments}

The [cmd.txt] parameter represents the commands file.

Commands File

A commands file is an ASCII text file that ends with a **.TXT**, **.VDX**, or no extension. The communications program attempts to open any text document as a commands file. You can create a commands file with a text editor or generate it from a custom application program. If the communications program encounters more than one commands file, it only opens and attempts to execute from the last one listed on the command line.

The commands file is a list of instructions to the communications program telling it what actions to perform. The communications program attempts to execute the commands sequentially from the top to the bottom of the file. If it encounters an error, it automatically stops executing and notifies the user of the error.

A commands file typically includes commands to:

- send and retrieve files from the memory card,
- send cross-reference files (**.TXT** files) to the memory card,
- receive the data file,
- change the ID on the LaserLite Mx,
- set the clock to the computer's time,
- resend the operating system software (**.OS** file), and so on.

It is important to list the commands in an order that matches an expected flow of operation for the LaserLite Mx. For example, the LaserLite Mx must be unlocked before it can accept any of the other commands in the list, so each commands file list will typically begin with an **I** (**Unlock**) command. Also, important data should be transferred from the LaserLite Mx prior to loading new operating system, application, or cross-reference files.

Commands File Commands

The following table summarizes the commands file's commands. The commands indicated by an asterisk (*) have been added or enhanced for use with LaserLite Mx and memory cards; additional information on these commands can be found on the pages indicated. Information on the other commands is in Chapter 1 of this manual.

Command	Description
`'`	Comment indicator.
`C` *<id>*	Set or change the unit's ID to the given ID. If no ID is given, the command fails.
`*D` *<file type> <filename>*	Delete file from memory card (page 134).
`*F` *[<output filename>]*	Display memory card's file management report (page 135).
`G`	Run the current application using the operating system **Go** command.
`I` *<id>*	Send an **Unlock** command to the LaserLite Mx using the given ID. The command is sent every 5 seconds for 35 seconds.
`*K`	Remove deleted files from memory card (page 134).
`L`	Send a **Lock** command and put the LaserLite Mx to sleep. When the unit is awakened with a keypress, it immediately runs the current application.
`M1` *<message>*	Set display line 1 to the given message.
`M2` *<message>*	Set display line 2 to the given message.
`*R` *[<# of bins>] <file type> <filename>*	Send file to LaserLite Mx (page 133).
`*S` *[<folder path>] <file type> <filename>*	Get file from LaserLite Mx (page 134).
`T`	Set the time on the LaserLite Mx to the computer's time (synchronize the clocks).
`Z`	Clear the data in the data collector.

Note: A space is required between the command and the arguments.

The communications program interprets the commands in the commands file in order. It issues the appropriate series of **P** commands (see pages 105–129 for information on the **P** command) to the operating system, which passes commands directly to the memory card module processor. The communications program uses the CRC checked commands and automatically adds the CRC to each command string.

Memory Card Commands for the Commands File

The following sections provide more information on the commands used in the **Vxcom** or **Download** commands file to communicate with a memory card. The following table lists the commands file commands that are used to communicate with the memory card. Note: A space is required between the command letter and the parameters. The following commands are described in detail on the following pages.

Command	Description
D *<file type> <filename>*	Delete file from card.
F *[<output filename>]*	File management report.
K	Remove deleted files.
R *[<number of bins>] <file type> <filename>*	Send file to card.
S *[<folder path>] <file type> <filename>*	Get file from card.

R Command

- The **R** command sends a file to the LaserLite Mx. If you use the *<file type>* argument, the file is sent to the memory card.

Syntax:

R *[<# of bins>] <file type> <filename>*

Parameters:

Argument	Description
# of bins	Modulus (size) for indexed (I) files only. Integer value can be in the range 1–65535. This argument is not used for sequential (S), identification (D), or boot (B) file types.
file type	The type of memory card file (either (I)ndexed, (S)equential, I(D)entification, or (B)oot).
filename	Name of file (include extension).

S Command

- The **S** command retrieves a file from the LaserLite Mx. If you use the *<file type>* argument, the file is retrieved from the memory card (instead of from RAM).

Syntax:

S *[<folder path>] <file type> <filename>*

Parameters:

Argument	Description
folder path	Name of folder that the retrieved file is put into; the file is placed in the current folder if a folder path is not given (optional argument).
file type	The type of memory card file (either (I)ndexed, (S)equential, I(D)entification, or (B)oot).
filename	Name of file (include extension).

D Command

- The **D** command deletes a file from the memory card.

Syntax:

D *<file type> <filename>*

Parameters:

Argument	Description
file type	The type of memory card file (either (I)ndexed, (S)equential, I(D)entification, or (B)oot).
filename	Name of file (include extension).

K Command

- The **K** command removes deleted files from the memory card.

Syntax:

K

F Command

- The **F** command requests the memory card's file management report. The operating system returns the data as it receives it from the memory card and stores it to a file that can be viewed and printed.

Syntax:

F *[<output filename>]*

Parameters:

Argument	Description
output filename	Name of file that the information is written to (optional argument). If a filename is not specified, the report is written to a file named **CFILES.TXT**.

If an <*output filename*> is provided, the file management report is written to the named file instead of to the default file **CFILES.TXT**.

The communications program presents the memory card's file management report in the following format:

```
Memory Card File and Memory Status Report

Total Space:         xx Kbytes
Bad Space:           xx Kbytes
Available Space:     xx Kbytes
Retrievable Space:   xx Kbytes

Number of deleted files:xx
Number of valid files:  xx

File #  Deleted    Type   Table Size  Kbytes   Name
1       No         B      0           26       BOOTCARD0
2       No         B      0           38       CPU
3       No         I      2203        1677     CRF1.TXT
…
```

File Types

The LaserLite Mx system supports the following types of files: *identification* (**D**), *boot* (**B**), *sequential* (**S**), and *indexed* (**I** or **H**). Each file type serves a different purpose. The file types are described in the following table.

Type	Purpose	Comments
D	A binary file that provides an ID for the memory card.	Only one (1) ID file may exist on the memory card at a time.
B	Two types of boot files serve two different purposes. One (CRD) is for the memory card operating software that executes in its 32K XRAM; the other (CPU) is the main operating system (***LMXxxx.OS***) and application of the LaserLite Mx.	These filenames are reserved by the LaserLite Mx system. CRD – The operating software for LaserLite Mx 32K XRAM. This file is required; it is created when the card is formatted using MXFORMAT.EXE. CPU – An operating system and application that may be booted from the memory card. This file is optional; see pages 145–146 for information on creating and loading a CPU file.
S	A binary file not intended for booting.	
I	This is the primary file type for data and cross-reference. Data is appended record-by-record. Each record is passed to the data management system to 'hash' and add to the record. This enables quick lookup.	A hash table size must be indicated when indexed files are created. Please see the notes on pages 141–142.
H	Another type of indexed file, but the decision of table entry is determined outside of the memory card data management system. This system may be useful for programs that must maintain a strict order of data, but require the ability to change or edit data.	**Vxcom** does not support transferring **H** files.

Commands File Examples

The following example demonstrates using a commands file with a LaserLite Mx:

```
I 0000000000
'Unlocks the unit to prepare it for communications
M1 Get File Report  'Send message to line 1 of unit.
F before.txt        'Get file report to write to before.txt.
M1 Downloading      'Send message to line 1 of unit.
M2 data.txt         'Send message to line 2 of unit.
S I data.txt        'Instruct unit to send an ‘I’ type file
                    'from its memory card.
M2 picklist.txt     'Send message to line 2 of unit.
S I picklist.txt    'Instructs unit to send another ‘I’
                    'type file from its memory card.
M1 Clearing files   'Send message to line 1 of unit.
M2 **************** 'Send message to line 2 of unit.
D I data.txt        'Mark ‘data.txt’ file as deleted.
D I picklist.txt    'Mark ‘picklist.txt’ file as deleted.
K                   'Erase deleted files from memory card
                    'to free up memory.
M1 Receiving new    'Send message to line 1 of unit.
M2 picklist file    'Send message to line 2 of unit.
R 2203 I picklist.txt 'Instruct unit to create a new ‘I’
                      'type file with 2203 ‘bins’ and to
                      'prepare to receive, index, and save
                      'each record.
M1 Set date and       'Send message to line 1 of unit.
M2 time               'Send message to line 2 of unit.
T                     'Synchronize clock with computer’s.
M1 Get File Report    'Send message to line 1 of unit.
F after.txt          'Get file report to write to after.txt.
L                     'Put LaserLite Mx unit to sleep.
```

The following commands file sends a cross-reference file named **crf1.txt** to the LaserLite Mx memory card:

```
I 0000000000
'Unlock the LaserLite Mx.
R 2203 I "crf1.txt"  'Send a file for the memory card
                     'to index to 2203 bins.
L                    'Lock the LaserLite Mx.
```

The following commands file sends a cross-reference file named **xref.txt** to the LaserLite Mx memory card:

```
I 0000000000
'Unlock the LaserLite Mx for communications.
D I "xref.txt"         'Delete previous copies of file.
K                      'Free up all available memory.
R 2203 I "xref.txt"    'Create and send file to the
                       'memory card, index to 2203 bins.
F                  'Create a file report to CFILES.TXT.
L                  'Lock the LaserLite Mx.
```

Vxcom or **Download** sends the file (one record at a time) to the LaserLite Mx memory card. When the LaserLite Mx receives the record, it indexes it and adds it to the data file.

To send additional files, add them to the commands file or create a new commands file, and execute **Vxcom** or **Download** again.

The following example commands file retrieves data from a file on a card, deletes the file from the card, sends a new image file to the card, then requests the file management report from the card.

```
I 0000000000
'Unlock the LaserLite Mx.
S I "data.txt"  'Receive a file, via YModem, from
                'the memory card.
D I "data.txt"  'Delete the data file just
                'transferred from the card.
K               'Remove deleted file from card.
R B "CPU"       'Send a new LaserLite Mx operating
                'system and boot program to the card.
F               'Request file management report.
G               'Start the program.
```

Note: When creating the commands file, do not include any characters on the **I** command line after the unit ID parameter . Any additional characters (including spaces) after the unit ID will cause this command to fail. In the above examples, the comments associated with the **I** commands are on the next line.

Notes on Communicating with a Memory Card

- All data transferred between the host and the memory card must begin with a command letter and end with a carriage return.
- The command letter may be lowercase or uppercase. When the command letter is uppercase, the memory card module software is in debug mode; when the command letter is lowercase, the software is in normal mode. The difference between debug mode and normal mode is that the communications between the host and memory card are CRC checked for normal mode, and not CRC checked for debug mode.
- For a lowercase key character command, the CRC is calculated from all of the bytes (including space and escape characters: & and ' or ") before the carriage return. The complements of the two CRC bytes are sent at the end of the command, before the carriage return, with the low byte sent first. The CRC of all the bytes sent across before the carriage return will always be **&B001**, if correct. The CRC of the returns from the card will only include the bytes before the carriage return.
- A command has two different types of fields besides the command letter: parameters to control the behavior of the command and data to be passed between the host and the memory card.
    - All of the fields for a command must be separated by a space and must be input in the order defined.
    - The parameters may be binary values, single byte ASCII, and ASCII strings (such as filename and key field).
    - A binary parameter may be up to 4 bytes long and can be input in both decimal and hexadecimal formats.
    - The data may be hexadecimal values (for binary data) or ASCII strings (for records).
    - The hexadecimal values must begin with an ampersand (&) and end with an ampersand, space, or carriage return.
    - ASCII strings (with more than one character) must be enclosed in either single ('*xxx*') or double ("*xxx*") quotation marks; a single non-numeric ASCII byte does not need to be enclosed in quotation marks.

- The returns from the card module can be status bytes (in decimal form), data (in hexadecimal or ASCII string format), or prompts responding to a single carriage input. The prompt for the firmware includes three bytes: a carriage return, a line feed, and a semicolon (:). The prompt for the memory card's boot program (Data Management System (DMS)) is: a carriage return, line feed, and greater-than sign (>).
- For a lowercase command letter, the CRC is calculated from all of the bytes (including space and escape characters: **&** and **'** or **"**) before the carriage return. The complements of the two CRC bytes are sent at the end of the command, before the carriage return, with the low byte first. The CRC of all the bytes sent across before the carriage return will always be **&B001**, if correct. The CRC of the returns from the memory card will only include the bytes before the carriage return.
- For indexed files, the DMS handles one record at a time. The fields within the record must be separated by a tab character.
- For sequential files, every byte in the data field of a command is written to the file.
- Doubling the "escape character" forces the character to be a data byte, for example, **"&&"** in a quoted string represents character **&**; **"''"** represents character **'**. If two continuous escape characters need to be used as escape (as when sending two strings together) a space must be used to separate the strings.
- Do not include a carriage return or a backspace in a quoted string. A carriage return can only be used as the end of transmission mark and the backspace key will delete the previous byte. If nonprintable bytes need to be sent as part of a string, they should be sent as a hexadecimal value.
- Any binary data, except the control characters (08 hex, 0D hex) and the escape characters (&, " or '), can be sent as ASCII strings.

About the Modulus

Notes about hashed indexes on the LaserLite Mx

Hash tables differ from other tables or arrays because they provide the LaserLite Mx system a fast way to seek data in an ASCII data file. They provide nonsequential access to data elements through the use of a hash function that converts the key field into an integer and divides it by the size (modulus) of the hash table. The remainder becomes the "key," "lookup value," or "bin" that indexes where to find the data element. The LaserLite Mx system provides a built-in hash function for distributing data elements into a hash table.

While the actual hash algorithm for the LaserLite Mx is written in 8051 assembler, it is based on the following C language code example which is based on Allen Holub's portable adaptation of Peter Weinberger's generic hashing algorithm.

```
/*---Hash PJW----------------------------------------
/*An adaptation of Peter Weinberger's (PJW) generic
/*hashing algorithm based on Allen Holub's version.
/*Accepts a pointer to a datum to be hashed and
/*returns an unsigned integer.
*-------------------------------------------------*/

#include <limits.h>

#define BITS_IN_int (sizeof(int) * CHAR_BIT)
#define THREE_QUARTERS ((int) ((BITS_IN_int * 3) / 4))
#define ONE_EIGHTH  ((int) (BITS_IN_int / 8))
#define HIGH_BITS   (~((unsigned int)(~0) >> ONE_EIGHTH))

unsigned int HashPJW (const char * datum)
{
   unsigned int hash_value, i;
   for (hash_value = 0; *datum; ++datum)
   {
      hash_value = (hash_value << ONE_EIGHTH) + *datum;
      if ((I = hash_value & HIGH_BITS) ! = 0)
         hash_value =
            (hash_value ^ (I >> THREE_QUARTERS 00 &
                ~HIGH_BITS;
   }
   return 9 hash_value);
}
```

What Modulus Should be Used When Creating an Indexed File?

The Videx hashing formula can accommodate any number of records, regardless of the modulus. Performance will decrease as the number of records becomes much larger than the modulus.

Use the following rules for selecting the modulus for LaserLite Mx indexed files:

- Optimal trade-off between performance and space efficiency is achieved when the anticipated number of records is less than twice the modulus.
- A prime number modulus provides the best distribution of data. The following prime numbers selections can provide optimal distribution:

  13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 21701, 23209, 44497.
- The LaserLite Mx can support a modulus up to 65535.

Common Precautions for Using a Memory Card

It is important to be aware of the following notes and precautions when using a memory card:

- Do not remove the memory card from the LaserLite Mx unless the lock switch is in the OFF position (towards the lock icon).
- Do not remove the memory card or turn off power to the LaserLite Mx during operations; such as, file transfer, card formatting, or file deletion. This could damage the memory card.

Doing either of the above could interrupt writing important data management information to the memory card and render some data irretrievable.

Note: Removing a memory card from the LaserLite Mx while the unit is ON automatically places the memory card module processor in ‘halt’ mode. This prevents any undesirable writing to the memory card. To restart the memory card module processor, the LaserLite Mx must restart from the ‘sleep’ mode. This happens when the unit is switched OFF or when it times out. ‘Sleep’ mode is indicated by a blank display.

The memory card is a precision electronic device:

- Do not apply pressure or shock.
- Do not bend or drop.
- Do not use or store the memory card in an environment subject to strong static electricity or electrical noise interference.
- Do not use the memory card in a hot, humid, or corrosive environment.
- Make sure no dirt or foreign particles are on the contact area. Do not touch the contact area with your fingers.
- Clean the memory card with a soft anti-static cloth (available at electronic stores).

Protecting the Data in the Memory Card

- Transferring files to and from the memory card is prevented when a write/protect adhesive seal is applied to the write/protect area. To enable file transfer, remove the write/protect adhesive seal.
- Apply the write/protect adhesive seal properly. Make sure that the write/protect adhesive seal is applied securely in its place.
- Data can be written to the memory card at least 250,000 times. For example, if data is written to the memory card 30 times a day, the card will last at least 20 years.
- Data can be lost or destroyed in the following situations:
    - Improper handling and use of memory card by the user or third party.
    - Exposure to static electricity or EMI (electronic magnetic interference).
    - Removing the memory card or turning off power to the LaserLite Mx during an operation.
    - The memory card can be used reliably over many years, but will eventually lose its ability to store and transfer data. At this point, replace with a new card.
    - We recommend saving your important data to another medium; such as, a floppy disk or hard disk.

Booting From the Memory Card

The LaserLite Mx system allows the developer to create and load bootable application files onto the memory card. When the LaserLite Mx system starts, it automatically checks for the presence of these files and boots under certain conditions.

To create the bootable file, use **Vxcom** as follows:

Vxcom*xxx* <*application name*>.s LMX*xxx*.os -f1 -k2

This creates a new image file called **FLASH.IMG**. Rename the **FLASH.IMG** file to **CPU**.

The file **LOADCPU.VDX** contains the following commands:

```
I 0000000000
'Unlock the LaserLite Mx for communications.
D B CPU  'Delete any previous copy of CPU boot file.†
K        'Free up all available memory.
R B CPU  'Load the CPU file from the computer.
F        'Create a file report to CFILES.TXT.
L        'Lock the LaserLite Mx.
```

† Note: If a CPU file is not on the memory card when the commands sequence is executed, the **D** command will not generate an error. This commands file works whether a CPU file is present or not.

You can use **LOADCPU.VDX** or create a similar commands file to send the new CPU file to the memory card. To do so, use this command line:

VXCOM*xxx* LOADCPU.VDX

This procedure assumes that an **OS** file is already installed and running on the LaserLite Mx. (Note: The LaserLite Mx **OS** filename is **LMX*xxx*.OS**, where ***xxx*** represents the version number of the file.) If the **.OS** file is not installed on the LaserLite Mx, you will need to load both the memory card and the main RAM with the desired program.

To load the new image file to both the memory card and the RAM on the LaserLite Mx, create the CPU file as described on the previous page, and use a text editor to modify **LOADCPU.VDX** as follows:

```
I 0000000000
'Unlock the LaserLite Mx for communications.
R        'Load files included on the command line.
D B CPU  'Delete any previous copy of CPU boot file.
K        'Free up all available memory.
R B CPU  'Load the CPU file from the computer.
F        'Create a file report to CFILES.TXT.
T        'Set LaserLite Mx time by computer’s time.
L        'Lock the LaserLite Mx.
```

Then use this command line:

Vxcom*xxx* <*application name*>.s LMX*xxx*.os LOADCPU.VDX

Sending Cross-Reference Files to the Memory Card

A commands file must be passed with **Vxcom** or **Download** to send cross-reference files to the memory card. Cross-reference files are typically used for lookups and data verification in applications built with Videx BASIC. The LaserLite Mx also permits applications to delete and add records to cross-reference files that reside on the memory card.

A cross-reference file for the memory card must be created as an ASCII text file and have a **.TXT** extension. The text file can be created with a text editor, a spreadsheet program, or exported from a database. A tab character must separate the fields and each record must end with a carriage return/line feed (which is the default behavior for computer-based text editors and ASCII exports from computer database programs). See page 149 for more information on creating a cross-reference file for a memory card.

Following is an example of a commands file to load a cross-reference file on the memory card:

```
I 0000000000
'Unlock the LaserLite Mx for communications.
D I <filename.txt> 'Delete any previous copies of the file.
K                  'Free up all available memory.
R 2203 I <filename.txt>    'Create and load new I type file.
F                  'Create a file report to CFILES.TXT.
L                  'Lock the LaserLite Mx.
```

When **Vxcom** or **Download** sends the file to the LaserLite Mx, it sends it one record at a time. As the LaserLite Mx receives the record, it indexes it and adds it to the data file.

To send additional files, add them to the commands file, or create a new commands file and execute **Vxcom** or **Download** again.

Transferring Data from the Memory Card

To retrieve data from the memory card, set up a commands file similar to this:

```
I 0000000000
'Unlock the LaserLite Mx for communications.
S I <filename.ext>'Open and send the named type and file.
L                 'Lock the LaserLite Mx.
```

This commands file does not clear the data from the file. To clear data, the file must be deleted with the line: **D *<file type> <filename>***.

Vxcom or **Download** appends the transferred data to a file of the same name if it exists; otherwise, it creates a new file.

Creating a Cross-Reference File for the Memory Card

You can use your computer to create a cross-reference file for the memory card. The file should be created as a tab-delimited ASCII text file with no quotes around the records or the fields. The structure of each record is as follows:

Data Field 1 (key field)	Tab	Data Field 2	Tab	Data Field 3	…

The maximum record length is 1000 characters. There is no limit to the number of fields within that 1000-character limit.

These are the steps to load a cross-reference file onto a memory card:

1. Create a tab-delimited ASCII file as defined above. In this example, we will use the name **PICKLIST.TXT** for the filename.

2. Using a text editor such as Windows Notepad, create a commands file that contains the following commands. Note: Do not include the comments.

```
I 0000000000          ' Unlock unit
D I picklist.txt      ' If file exists on card, delete it
K                     ' Free all available memory on card
R 2203 I picklist.txt ' Load file to memory card
F                     ' Write memory card file system
                      ' status to CFILES.TXT
G                     ' Restart application
```

Opening a File on the Memory Card

Up to 60 different files may reside on the memory card including deleted files. In Videx BASIC, the following command syntax opens an existing file named **picklist.txt** on the memory card:

```
CARDCMD O, I, picklist.txt
```

The memory card software system receives the command, processes it, then this syntax retrieves the file handle.

```
cardresult% = CARDSTATUS(cardreturn$)
```

Cardresult% captures the success or failure of the command. If **CARDSTATUS** returns 0 to **cardresult%**, then **CARDCMD** executed successfully. If **picklist.txt** is not on the memory card, then **cardresult%** should have a value of 31.

If **picklist.txt** is found and opened, then **cardreturn$** will contain the file handle number. You can then put the file handle number into a variable…

```
IF cardresult% = 0 THEN picklist$ = cardreturn$
```

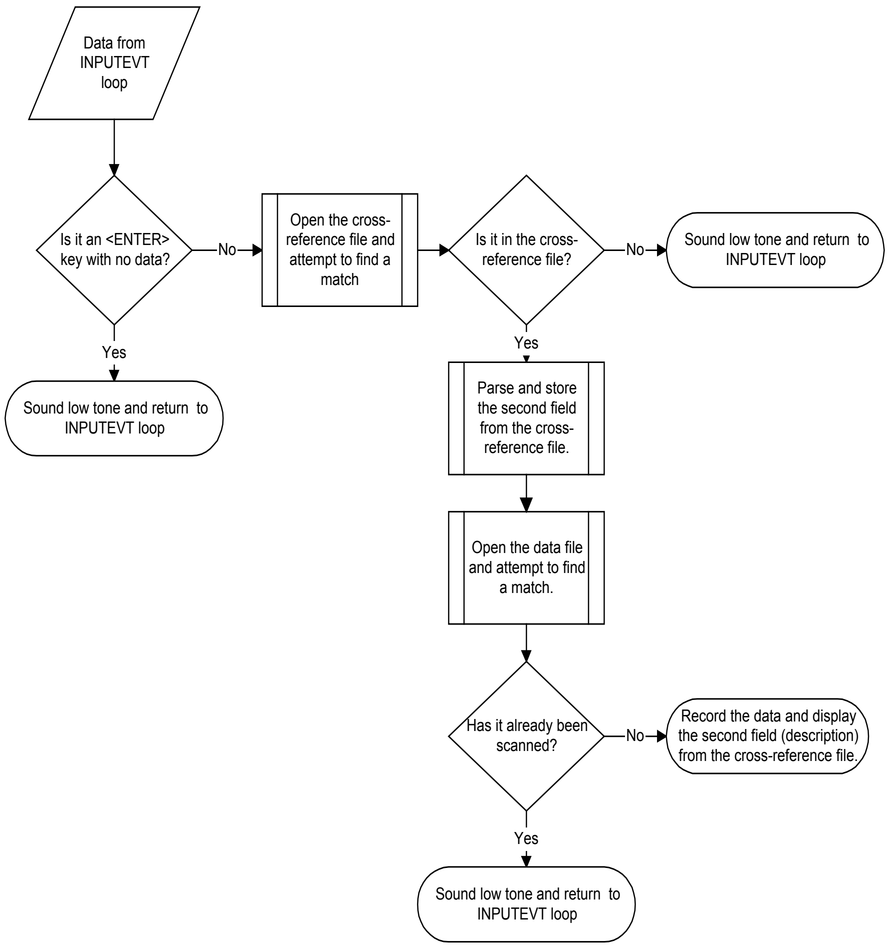
…then later re-open that file by referencing the file handle number only.

```
fileopen$ = “O, “ + picklist$
CARDCMD fileopen$
```

See the *Videx BASIC Manual* for information on Videx BASIC.

Data and Cross-Reference File Handling on the Memory Card

The following routine demonstrates matching input against a cross-reference file, extracting data from the second field of the record, checking for duplicate scans in the data file, and saving data. It processes input from an **INPUTEVT** loop as follows:


Data from INPUTEVT loop
Is it an <ENTER> key with no data?
No
Yes
Sound low tone and return to INPUTEVT loop
Open the cross-reference file and attempt to find a match
Is it in the cross-reference file?
No
Sound low tone and return to INPUTEVT loop
Yes
Parse and store the second field from the cross-reference file.
Open the data file and attempt to find a match.
Has it already been scanned?
No
Record the data and display the second field (description) from the cross-reference file.
Yes
Sound low tone and return to INPUTEVT loop

This is the corresponding BASIC source code.

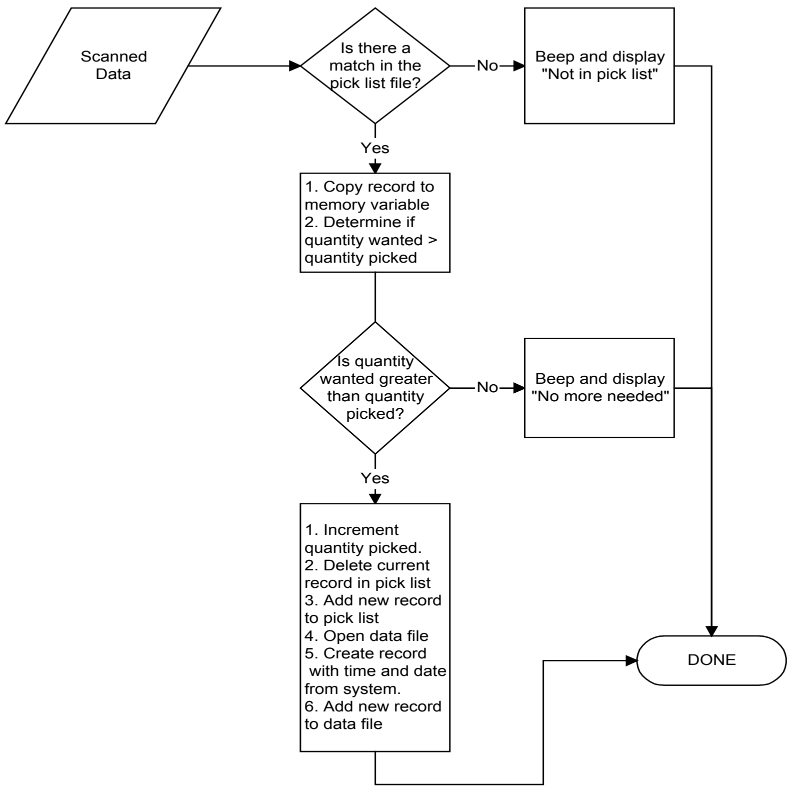
```
fn_input:
'*   fn_input
'*  description: Respond to a user input event.
'*  Data is validated by checking if it is in cross-reference
'*  file on memory card. If data is valid, emit beep, flash Valid
'*  Scan LED, and display description field corresponding to the
'*  key field which was scanned.

   if device% = 1 then
       if len(data$) = 0 then
         'it's an ENTER key, with no data
         sound 698, 250  'sound a low beep
         return
       endif
   endif

   cardcmd O, 2203, I, "bigcrf.txt"    'open crf file on card
   gosub process_cardcmd_error
   cardresult% = cardstatus (cardreturn$)
   gosub process_cardret_error

   cardcmd F, data$       'search for data$ in crf file
   gosub process_cardcmd_error
   cardresult% = cardstatus (cardreturn$)
   gosub process_cardret_error

   if (cardresult% <> 35) then   'data found in crf file
      tabpos% = instr (cardreturn$, chr$(9))  'look for tab
      desc$ = mid$(cardreturn$, (tabpos%+1),(len(cardreturn$)-tabpos%))
      cardcmd O, 2203, I, "data.txt"   'open data file on card
      gosub process_cardcmd_error
      cardresult% = cardstatus (cardreturn$)
      gosub process_cardret_error
      cardcmd F, data$         'search for data$ in data file
      gosub process_cardcmd_error
      cardresult% = cardstatus (cardreturn$)
      gosub process_cardret_error

   if (cardresult% = 35) then  'data not already in data file
         option(258) = 1       'turn on the LED
         sound 1446, 250       'good beep
         option(258) = 0       'turn off the LED
         cardcmd A, data$ 'add this scan to data file on card
         gosub process_cardcmd_error
         cardresult% = cardstatus (cardreturn$)
         gosub process_cardret_error
         display$ = desc$      'display the input on the screen
         mode = mode_dn%       'scroll down to last line of file
         full_display$ = desc$  'initialize scroll right & left
         shift% = 0
      else
         gosub bad_tone        'duplicate scan, sound bad beep
      endif
    else
        gosub bad_tone         'not in crf file, sound bad tone
    endif
return
```

Changing Records in Memory Card Files

Because of the characteristics of the flash memory in memory cards, changing a record on the card entails the following steps:

1. Read the desired record into a memory variable.
2. Make the desired changes to the data in the variable.
3. Delete (hide) the old record.
4. Add a new record with the contents of the memory variable.

This routine demonstrates editing "on-the-go." The **picklist.txt** file loaded into the LaserLite Mx is a pick list. Each record has four fields delimited by tabs. The fields are 1) data, 2) description, 3) quantity wanted, and 4) quantity picked. Following is a flowchart that describes the flow of the validation:


Scanned Data
Is there a match in the pick list file?
No
Beep and display "Not in pick list"
Yes
1. Copy record to memory variable
2. Determine if quantity wanted > quantity picked
Is quantity wanted greater than quantity picked?
No
Beep and display "No more needed"
Yes
1. Increment quantity picked.
2. Delete current record in pick list
3. Add new record to pick list
4. Open data file
5. Create record with time and date from system.
6. Add new record to data file
DONE

The user scans one item at a time until each item in the list is picked. The following program is the corresponding Videx BASIC code for the input routine. (Note: The following program contains program lines that exceed the width of the page; so we will use the pipe character (|) to indicate that the program line continues.)

```
fn_input:
  if device% = 1 then
     if len(data$) = 0 then
                         'it's an ENTER key, with no data
      sound 698, 250     'sound a low beep
      return
     endif
  endif

  gosub open_picklist       'open the pick list file
  cardcmd F, data$          'search for data$ in pick file
  gosub process_cardcmd_error
  cardresult% = cardstatus(cardreturn$)

  if (cardresult% = 35) then 'data not found in data file
    cls                     'or if not found in pick list,
                            'do not accept it
    locate 0,0
    print data$
    print "Not in pick list";
    gosub card_error_tone
    sleep 2500
  elseif (cardresult% <> 0) then
    gosub process_card_error
  else
    cardcmd M      'copy the record to a memory variable
    gosub process_cardcmd_error   'so it can be edited.
    cardresult% = cardstatus(cardreturn$)'sort the fields
    tabpos1% = instr (cardreturn$, chr$(9)) 'look for tab
    tabpos2% = instr (tabpos1% + 1, cardreturn$, chr$(9))
    desc$ = mid$(cardreturn$, (tabpos1% + 1), (tabpos2%)|
(tabpos1% + 1)))

    tabpos3% = instr (tabpos2% + 1, cardreturn$, chr$(9))

    qtywanted% = val(mid$(cardreturn$, (tabpos2% + 1),|
((tabpos3%+1) - tabpos2%)))
    qtypicked% = val(mid$(cardreturn$, (tabpos3% + 1),|
(len(cardreturn$) - tabpos3%)))
```

```
      IF qtypicked% >= qtywanted% THEN
         cls
         locate 0,0
         print desc$
         print "No more needed!";
         gosub card_error_tone
         sleep 2500
         gosub open_datafile
      ELSE
         cardcmd M,H    'delete record at the move pointer
         gosub process_cardcmd_error
         cardresult% = cardstatus(cardreturn$)
         gosub process_card_error
                        'create a new record

         option(258) = 1            'turn on the LED
         sound 1446, 250            'good beep
         option(258) = 0            'turn off the LED
         record$ = data$ + chr$(9) + desc$ + chr$(9) +|
str$(qtywanted%) + chr$(9) + str$(qtypicked%+1)
         cardcmd A, record$         'add the new record
         gosub process_cardcmd_error
         cardresult% = cardstatus(cardreturn$)
         gosub process_card_error
         display$ = desc$       'display description on screen

         gosub open_datafile

         record$ = data$ + chr$(9) + date$() + chr$(9) + time$()

         cardcmd A, record$
         gosub process_cardcmd_error
         cardresult% = cardstatus(cardreturn$)
         gosub process_card_error

         lastinput$ = data$

      ENDIF

      mode = mode_dn%     'same as if just scrolled down to
                          'last line of the file
      full_display$ = desc$  'initialize to scroll right & left
      shift% = 0

   endif

return
```

LaserLite Mx Error Handling Strategies

The LaserLite Mx operating system and memory card software system provide a comprehensive set of error reporting codes and conditions.

The errors may be reported in any of three ways:

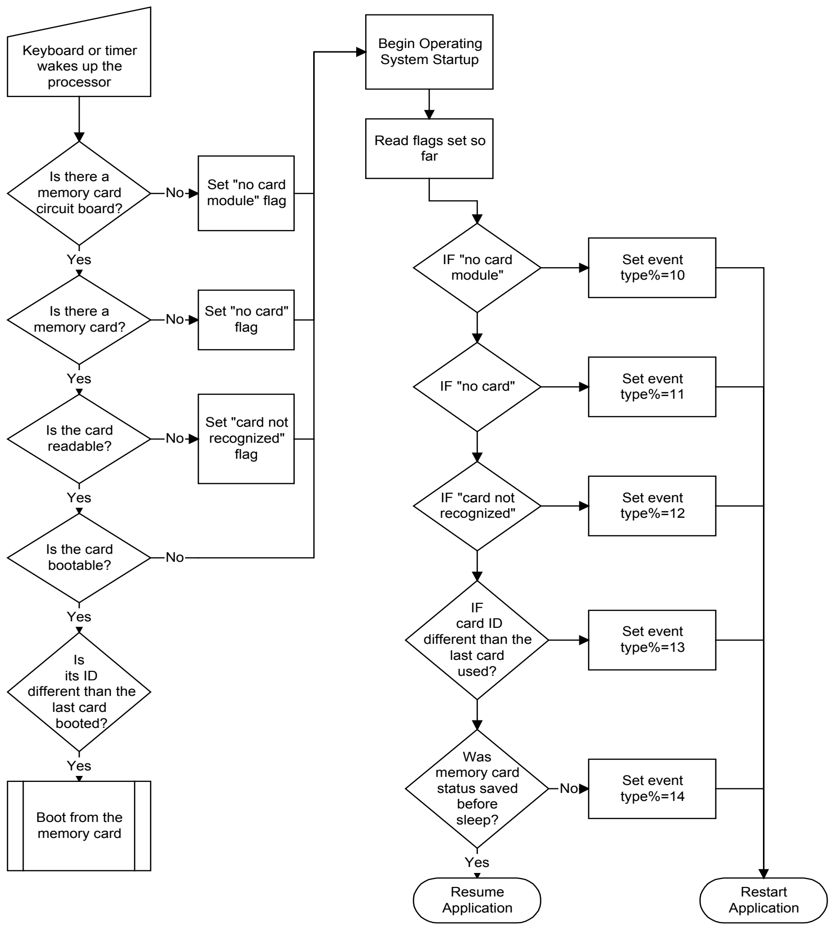
1. By the LaserLite Mx operating system through events reported by the Videx BASIC **INPUTEVT** statement.
2. By the Videx BASIC **CARDCMD** statement as global errors.
3. By the Videx BASIC **CARDSTATUS** function as integer return values.

Examples of error reporting are shown on the following pages. See the *Videx BASIC Manual* for information on the **INPUTEVT** statement, **CARDCMD** statement, and the **CARDSTATUS** function.

Operating System Error Handling

The LaserLite Mx operating system checks for error conditions both when it starts an application and when it returns from sleep (0 to 599).

The following flowchart illustrates the sequence of steps:



Keyboard or timer wakes up the processor
Is there a memory card circuit board?
No
Set "no card module" flag
Yes
Is there a memory card?
No
Set "no card" flag
Yes
Is the card readable?
No
Set "card not recognized" flag
Yes
Is the card bootable?
No
Yes
Is its ID different than the last card booted?
Yes
Boot from the memory card
Begin Operating System Startup
Read flags set so far
IF "no card module"
Set event type%=10
IF "no card"
Set event type%=11
IF "card not recognized"
Set event type%=12
IF card ID different than the last card used?
Set event type%=13
Was memory card status saved before sleep?
No
Set event type%=14
Yes
Resume Application
Restart Application

As illustrated, any of the error conditions will cause the operating system to automatically restart an application and set the appropriate event flag.

The following two BASIC routines are called by the Videx BASIC **INPUTEVT** statement in response to event type% = 13 and event type% = 14. They are found in the **MX-DEMO.B** sample application.

```
evt_card_interrupted:
'*
'*  evt_card_interrupted
'* description: Since the sleep routine saves the latest
'* status from the memory card software system, it can then
'* update the card on the next CARDCMD issued after waking
'* from sleep. If there is no valid status available, it
'* indicates that an event like a power failure or something
'* interrupted the process the status could be saved.
'* In either case, the application will automatically be
'* restarted. This routine notifies the user.
'*
       cls
       locate 0,0
       print "Communication to"
       print "card interrupted";
       sleep 2000
       cls
       locate 0,0
       print "Application"
       print " restarted ";
       sleep 2000
       cls
return

evt_cardID:
'*
'*  evt_cardID
'* description: Since the last time the OS ran, the card ID
'* has changed. The OS automatically restarts the
'* application and sets this event. This routine notifies
'* the user.
'*
       cls
       locate 0,0
       print "Card ID changed"
       sleep 2000
       cls
       locate 0,0
       print "Application"
       print " restarted ";
       sleep 2000
       cls
return
```

CARDCMD Statement Errors

The Videx BASIC **CARDCMD** statement may generate global errors that indicate that the operating system was unable to send the statement to the memory card software system. It can generate the following global errors:

Error	Description	Possible Cause
-1	Communications with the memory card timed out.	The memory card software system is busy processing the last command set it received or it didn't have enough time to finish processing the current command. In either case, try using the **CARDCMD** statement with the [!] option on commands that require more time than 800ms to process.
-2	A memory card module was not detected at startup.	**CARDCMD** reads the system startup flags before attempting to communicate with the memory card software system.
-3	A memory card was not detected at startup.	If a memory card is inserted, the system should be restarted to properly set the flags that a card is present.
-4	The memory card does not have a recognized format.	Unformatted card.
-11	The memory card processor is asleep and cannot receive a command.	Removing the memory card without first turning off the LaserLite Mx can cause this error.

The following BASIC routines handle the global errors from **CARDCMD** and are included in the **MX-DEMO.B** source code:

```
process_cardcmd_error:
'*
'*  process_cardcmd_error
'* description: Look for and report error with CARDCMD.
'*

  crderr% = ERR
  IF crderr%  = 0 THEN         'no error

  ELSEIF crderr% = -11 THEN    'Timeout error; perhaps card
    cls                        'was pulled out or the memory
    locate 0,0                 'card was busy when CARDCMD
    print "No response from"   'tried to issue a command
    print " memory card! ";    'Remember, pulling out the
      gosub card_error_tone    'card freezes the memory card
    sleep 1                    'processor. It must be reset
                               'for it to restart. That's why
                               'it's best to go to sleep and
                               'let the system bring it back
                               'up when it wakes up.

  ELSEIF crderr% = -2 THEN
    goto evt_nomodule

  ELSEIF crderr% = -3 THEN
    goto evt_nocard_cmd

  ELSEIF crderr% = -4 THEN
    goto evt_badcard

  ELSE
    cls
    locate 0,0
    print " Cardcmd error "
    print " # " + str$(crderr%) ;
    gosub card_error_tone
    sleep 2500
    running = false%
  ENDIF
return
```

```
evt_nomodule:
'*
'*  evt_nomodule
'* description: This event is set when the system starts and
'* the monitor determines that there is no response from a
'* memory card module. Since this application is designed to
'* run with a memory card, it should not be allowed to run
'* when this event is encountered.
'*

   cls          'A -2 error is set by the system at start up
   locate 0,0   'if it determines there is no memory card
   print " No card module"  'module present. This program
                            'doesn't do much good on a
                            'system like that.
   print "    detected   ";
   gosub card_error_tone
   sleep 3000
   running% = false%
   end

return


evt_nocard:
'*
'*  evt_nocard
'* description: This event is set when the system starts and
'* the monitor determines that no memory card is inserted.
'* Since this application is designed to run with a memory
'* card, it should not be allowed to run when this event is
'* encountered. It calls the sleep routine to allow the user
'* to insert a card and then restart the application.
'*

             'After waking up from sleep, the system
             'determined that there was no card inserted
      cls    'and set a global flag to indicate no card.
      locate 0,0  'The system must wake up from sleep with a
      print "    No card     "   'memory card inserted to
      print "   inserted!    ";  'clear this flag.
      gosub card_error_tone
      sleep 2500
      cls
      locate 0,0
      print "Insert card then"
      print "  press a key   ";
      sleep 10

return
```

```
evt_nocard_cmd:
'*
'*  evt_nocard_cmd
'* description: This routine is called when a -3 (no card)
'* error is received from CARDCMD. It warns the user then
'* exits to the OS. When the memory card is re-inserted,
'* then the user may press and hold the scan button. The
'* application restart allows the monitor to determine if a
'* card has been placed in the slot. Since this application
'* is designed to run with a memory card, it should not be
'* allowed to run without one inserted.
'*

     cls
     locate 0,0
     print "    No card     "
     print "   inserted!    ";
     gosub card_error_tone
     sleep 2000
     cls
     locate 0,0
     print "Insert card then"
     print "hold SCAN button";
     sleep 5000
     running% = false%

return
```

```
evt_badcard:
'*
'*  evt_badcard
'* description: This event occurs when the system cannot
'* recognize the memory card or the card won't boot. If the
'* memory card isn't formatted, then the system sets a
'* global flag to indicate that the memory card was not
'* recognized. On the other hand, a partially formatted card
'* may not give the system any problem until the application
'* attempts to send it a command. Then this routine may be
'* called because of an error 05 indicating that the card
'* software system didn't boot.
'*

     cls
     locate 0,0
     print "    Card not    "
     print "   recognized   ";
     gosub card_error_tone
     sleep 2000
             cls
             locate 0,0
             print " Reformat card  "
             print "  then retry    ";
             sleep 2000
             cls
             locate 0,0
             print "    running     "
             print "  application   ";
             sleep 3000
             cls
             locate 0,0
             print " Exiting to OS  "
             print "                ";
             sleep 1500
             running% = false%

return
```

CARDSTATUS Function Errors

The final type of error is generated by the memory card software system itself. The Videx BASIC **CARDSTATUS** function returns either an integer or a string value representing data. When the value of the **CARDSTATUS** integer return equals 0, then the given operation is considered successful and the string returned by **CARDSTATUS** represents data.

Different commands may return different integer errors, some of which may be part of normal operations. For example, when moving the record pointer forward or backward in an indexed file, a return of 36 indicates the end of file and 37 indicates the beginning of file. In the same circumstances, a return of 23 indicates that the file contains no data.

The variety of possible commands and error routines requires careful consideration of whether the error has been generated by user error or coding error. In any case, for reliable operations, it is important to check the integer return of each call to **CARDSTATUS**. See the *Videx BASIC Manual* for complete information on **CARDSTATUS**.

CARDSTATUS can return the following integer system errors:

Error	Description	Note
-1	Communications with the memory card timed out.	The memory card software system does not respond.
-2	A memory card module was not detected at startup.	As with the **CARDCMD** statement, **CARDSTATUS** first checks the system flags before attempting to communicate with the memory card module. Errors -2 and -3 will be returned if the system flags indicate.
-3	A memory card was not detected in the module at startup.	
-5	The CRC of the return from the memory card system does not match the one calculated by **CARDSTATUS**.	System communications problem between the main CPU and the memory card module.
-6	The return from the memory card is unrecognized.	Same as above.
-7	The designated string is too short for return from the memory card.	Dimension the string memory variable that receives data from **CARDSTATUS** to a larger value.
-8	No **CARDCMD** was issued since the last call to **CARDSTATUS**.	This indicates a coding error. The system expects **CARDCMD** and **CARDSTATUS** to be called in pairs.
-10	Card is still busy when **CARDSTATUS** called.	The memory card software system is still processing an instruction.
-11	Card is asleep when **CARDSTATUS** called.	Removing the memory card during operations can cause this error.
-15	The return from the memory card has exceeded the 2K buffer.	This should occur only with 1K data records containing '&' as every 2nd character.

The following BASIC routines handle integer errors from **CARDSTATUS** and are included in the **MX-DEMO.B** source code:

```
process_card_error:
'*
'*  process_card_error
'* description: Since the CARDSTATUS() function returns an
'* integer error code, it can be quickly analyzed and
'* processed with each call.
'*


   IF crderr% THEN
                'already handled by cardcmd error processing
   ELSEIF (cardresult% = 0)THEN  'No error
   ELSEIF (cardresult% = 5)THEN  'This response can only
        error5% = true%          'come from firmware. It
        n_times% = 1             'indicates the DMS software
        cls                      'isn't booted, perhaps
        locate 0,0               'because the DMS boot file
        print " Card system  "   'is not on the card.
        print "    error!    ";
        goto boot_card

   ELSEIF (cardresult% = 32) THEN
                          'This error means no file is open
        cls
        locate 0,0
        print "  No file open! "
        print "                ";
        gosub card_error_tone
        return

   ELSEIF  (cardresult% = 37)       'top of file
        OR (cardresult% = 36)       'bottom of file
        OR (cardresult% = 23) THEN  'no data

       ELSE

error_ring:         gosub card_error_tone
                    'otherwise just report the error number
                    cls
                    locate 0,0
                    print "Crd return error"
                    print "# " + str$(cardresult%);
                    sleep 3000
       ENDIF

return
```

```
boot_card:
'*
'*  boot_card
'* description: This routine attempts to load and run the
'* DMS software on the memory card.
'*

   cardcmd N         'Initialize the card.
   cardcmd B         'Send the boot command to start the DMS
   cardresult% = cardstatus(buffer_var$)
                     'software in the card.
   cls

   IF (cardresult% = 31) THEN
      goto evt_no_boot

   ELSEIF (cardresult% > 0) THEN    'If it still didn't boot,
      goto evt_badcard              'then perhaps the card
                                    'isn't formatted.

   ELSE
      gosub evt_card_interrupted
                     'If the boot was successful,
      sleep 1        'then something caused it to fail in the
   ENDIF             'first place. As a safeguard, the
                     'application should be restarted to
                     're-initialize pointers.

return
```

```
evt_no_boot:
'*
'*  evt_no_boot
'* description: This event occurs when the memory card won't
'* boot. A partially formatted memory card may not give the
'* system any problem until the application attempts to send
'* it a command. Then this routine may be called because of
'* an error 05 indicating that the card software system did
'* not boot and a subsequent attempt to boot it failed.
'*
     cls
     locate 0,0
     print " Memory card has "
     print " no boot file!  ";
     gosub card_error_tone
     sleep 2000
     cls
     locate 0,0
     print " Reformat card  "
     print "  then retry   ";
     sleep 2000
     cls
     locate 0,0
     print "    running     "
     print "  application   ";
     sleep 2000
     cls
     locate 0,0
     print " Exiting to OS  "
     print "                ";
     sleep 1300
     running% = false%
return

card_error_tone:
'*
'*  card_error_tone
'* description: Tone indicating a card command error.
'*

     sound 2349, 300
     sound 1885, 600
return
```

Chapter 6

DuraTrax, LaserLite, LaserLite Pro and LaserLite Mx Data Files

This chapter contains information on:

- The default data file.
- The **Scan.S** data file.
- The TimeWand II-style data file (**Timewand.src**).

Default Data File

The default data file for the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx is similar to Figure 6-1.

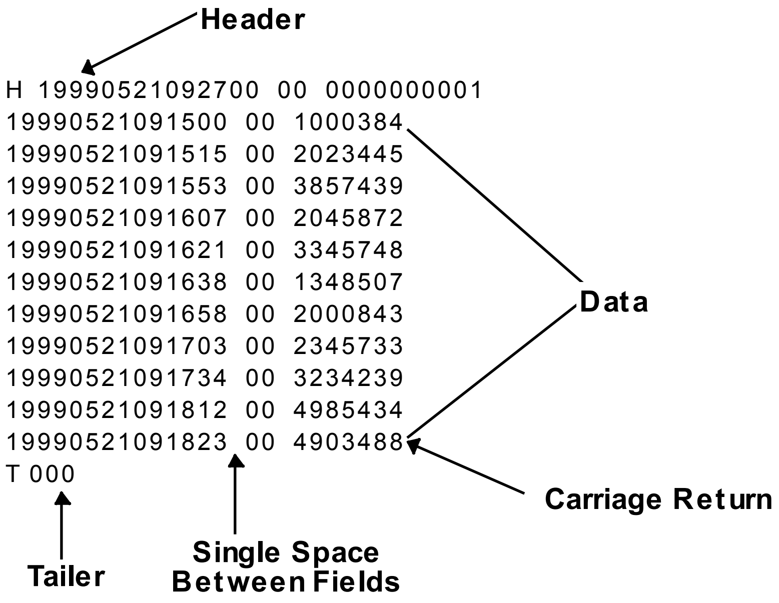
```
10-28-99   09:34:28   00002169
10-28-99   09:40:15   00002378
10-28-99   09:54:55   01:00000004859F
10-28-99   10:22:12   87965
10-28-99   10:37:32   ABC-123
10-28-99   14:56:32   00003567
```

Figure 6-1 Default Data File

The default data file lists the date of the entry, followed by the time of the entry, followed by the bar code data, Touch Memory button's serial number, or keypad entry.

Scan.s Data File

Data transferred from a DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx programmed with the **Scan.s** application program, is saved as a TimeWand I/DuraWand-style raw scan file. The raw scan file can be viewed using a text editor. This file has not been formatted by a database program or by a file manager program. Figure 6-2 shows data in a raw scan file from transferring a single data collector programmed with **Scan.s**.



```
H 19990521092700 00 0000000001
19990521091500 00 1000384
19990521091515 00 2023445
19990521091553 00 3857439
19990521091607 00 2045872
19990521091621 00 3345748
19990521091638 00 1348507
19990521091658 00 2000843
19990521091703 00 2345733
19990521091734 00 3234239
19990521091812 00 4985434
19990521091823 00 4903488
T 000
```

Figure 6-2 TimeWand I/DuraWand-Style Raw Scan File Format

The Header always appears as the first line in the scan file. The Header lists the unit's ID and the date and time of the data transfer.

The Tailer marks the end of the transferred data and identifies any errors in the transferring process. Each line of data between the Header and the Tailer represents individual bar code scans, keypad entries, or button reads. There is a single space character between each field in the raw scan file, and a carriage return ends each line.

The Header

Following are the definitions for the various parts of the Header.

1. The Header always begins with a capital **H** before the line of data. It indicates the beginning of the data transferred from a single DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx.

   H 19990521092700 00 0000000001

2. This indicates the year the data file was transferred.

   H **1999**0521092700 00 0000000001

3. This indicates the month and the day the data file was transferred.

   H 1999**0521**092700 00 0000000001

4. This uses 24-hour time to indicate the hour, minutes, and seconds that the data file was transferred.

   H 19990521**092700** 00 0000000001

5. This is the source code. The source code is currently undefined in the Header but may be used in the future.

   H 19990521092700 **00** 0000000001

6. This is the data collector’s ID.

   H 19990521092700 00 **0000000001**

The Data

Following are the definitions for the various parts of the data.

1. This indicates the year the data was entered.

   19990521091500 00 1000384

2 This indicates the month and the day the data was entered.

   1999**0521**091500 00 1000384

3. This uses 24-hour time to indicate the hour, minutes, and seconds that the data was entered.

   19990521**091500** 00 1000384

4. This number represents the origin of data. This number indicates the bar code symbology. For example, an origin of data of 00 indicates that the data was collected from scanning a whole Code 3 of 9 bar code; a 01 indicates that the data was collected using a scanpad or keypad; a 02 is a UPC-A or UPC-E bar code, and so on. See page 180 for a list of the origin of data codes.

   19990521091500 **00** 1000384

5. This is the data; it can be bar code data, a Touch Memory button serial number, or a keypad entry. The data can vary in length and can include spaces. (Note: A Touch Memory button's serial number is always a 12-digit hexadecimal number.)

   19990521091500 00 **1000384**

6. This portion of the data line is fixed in length.

   19990521091500 00 1000384

The Tailer

Following are the definitions for the various parts of the Tailer.

1. The Tailer always begins with a capital **T** before the line of data. It indicates the end of the data transferred from a single DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx.

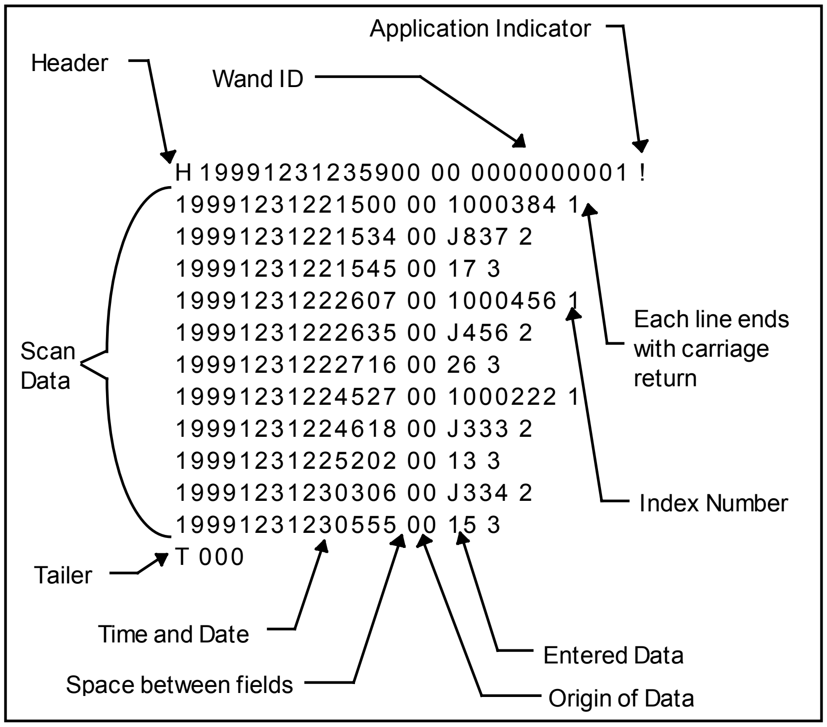
   <u>T</u>000

2. These three digits identify any errors that may have been detected in the data transfer. If these are all zeros, this means there were no errors.

   T**<u>000</u>**

TimeWand II-Style Data File (Timewand.src)

Data transferred from a DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx using the **Timewand.src** template, is saved as a TimeWand II-style data file. A TimeWand II-style data file is almost identical to the raw scan file, but it also contains application-indexing information. The data file can be viewed using a text editor. This data file has not been formatted by a database program or by a file manager program. Figure 6-3 shows data in a TimeWand II-style data file from transferring a single data collector programmed using **Timewand.src**.



```
H 19991231235900 00 0000000001 !
19991231221500 00 1000384 1
19991231221534 00 J837 2
19991231221545 00 17 3
19991231222607 00 1000456 1
19991231222635 00 J456 2
19991231222716 00 26 3
19991231224527 00 1000222 1
19991231224618 00 J333 2
19991231225202 00 13 3
19991231230306 00 J334 2
19991231230555 00 15 3
T 000
```

Figure 6-3 Timewand.src TimeWand II-Style Data File

The Header appears as the first line in the data file. The Header identifies the unit and the time of the transfer. The exclamation mark (!) after the Header shows that the unit's data is indexed. Each line of data ends with a space followed by the index number. The index number indicates which Input Handler the data belongs to; this indexing information is useful when separating the data into fields. (Note: Input Handlers are numbered from top to bottom and from left to right.)

For example, Figure 6-4 shows a sample application. (Note: The number in each Input Handler's name shows the Input Handler's ID; the IDs are numbered from top to bottom and from left to right.) Figure 6-5 shows a corresponding data file. Notice that the index number matches the Input Handler number.

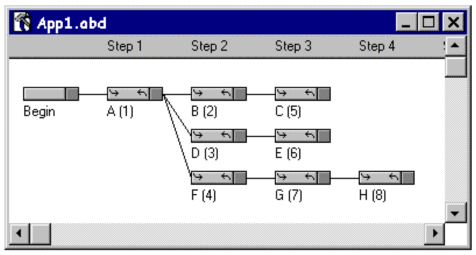
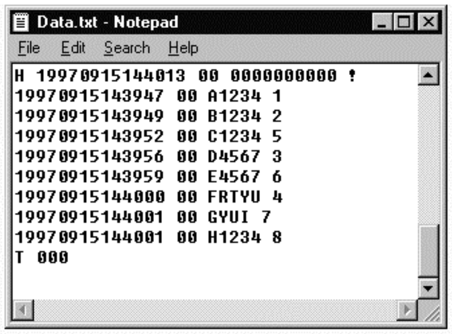


Figure 6-4 Sample Application



```
H 19970915144013 00 0000000000 !
19970915143947 00 A1234 1
19970915143949 00 B1234 2
19970915143952 00 C1234 5
19970915143956 00 D4567 3
19970915143959 00 E4567 6
19970915144000 00 FRTYU 4
19970915144001 00 GYUI 7
19970915144001 00 H1234 8
T 000
```

Figure 6-5 Sample Application Data File

The Tailer marks the end of the transferred data. Each line of data between the Header and the Tailer represents individual data entries.

The data fields are separated by spaces, and each line of the data file ends with a carriage return and line feed.

The Header

This section defines the highlighted portions of the Header.

1. The Header begins with a capital **H** before the line of data. It indicates the beginning of the data.

   H 19991231235900 00 0000000001 !

2. This indicates the year the data file was transferred.

   H **1999**1231235900 00 0000000001 !

3. This indicates the month and the day the data file was transferred.

   H 1999**1231**235900 00 0000000001 !

4. This uses 24-hour time to indicate the hour, minutes, and seconds the data file was transferred.

   H 19991231**235900** 00 0000000001 !

5. This is the source code. The code is currently undefined in the Header but may be used in the future.

   H 19991231235900 **00** 0000000001 !

6. This is the identification number belonging to the unit. This is always a ten-digit number.

   H 19991231235900 00 **0000000001** !

7. This is the application indicator; it appears at the end of the Header if the data has been indexed.

   H 19991231235900 00 0000000001 **!**

The Data

This section defines the highlighted portions of the data.

1. This indicates the year the data was entered.

   19991231221500 00 1000384 1

2. This indicates the month and the day the data was entered.

   1999**1231**221500 00 1000384 1

3. This indicates the hour, minutes, and seconds (24-hour) that the data was entered.

   19991231**221500** 00 1000384 1

4. This number represents the origin of data and indicates the type of data entered. Bar code symbologies and Touch Memory buttons have their own origin of data numbers. For example, 00 indicates that the data was collected by scanning a complete Code 3 of 9 bar code, a 01 indicates that the data was collected using either a bar code scanpad or the keypad, 02 indicates that the data was collected from scanning a UPC bar code, and so on. See page 180 for a list of the origin of data numbers.

   19991231221500 **00** 1000384 1

5. This is the actual data; it can be from a bar code scan, keypad entry, or Touch Memory button touch. The data can vary in length and may include spaces.

   19991231221500 00 **1000384** 1

6. Each data line will end with an index number. The index number corresponds to the Input Handler's ID.

   19991231221500 00 1000384 **1**

7. This portion of the data line is fixed in length.

   19991231221500 00 1000384 1

The Tailer

The Tailer always begins with a capital **T** before the line of data. It indicates the end of the data transferred from a single data collector.

```
T  000
```

Origin of Data

The following table defines the origin of data for the following bar code symbologies:

Bar Code Symbology	Origin of Data
Complete Code 3 of 9 bar code	00
Keypad or bar code scanpad entry	01
UPC A or E bar code	02
Interleaved 2 of 5 bar code	03
Codabar bar code	04
EAN/JAN bar code	06
Code 128 bar code	10

The following table shows the family code and the origin of data for some of the Dallas Semiconductor Touch Memory buttons:

Button	Family Code (Hexadecimal)	Origin of Data
DS1990A	01	31
DS1991	02	32
DS1992	08	38
DS1993	06	36
DS1994	04	34
DS1995	0A	40
DS1996	0C	42
DS1982	09	39
DS1985	0B	41
DS1986	0F	45
DS1920	10	46

The origin of data for a Touch Memory button is derived by taking the hexadecimal family code number of the button, converting it to a decimal number, and adding 30.

For example: The DS1920 button has a family code of 10 hexadecimal, which is equivalent to 16 decimal; add 30 and you get 46, which is the origin of data of a DS1920 button.

Chapter 7

Cross-Reference File Convert Programs

This chapter contains information on:

- Cross-reference files for DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.
- The **Vxcrfw.exe** cross-reference file convert program for Windows.
- The **Vxcrf.exe** cross-reference file convert program for DOS.
- Using a cross-reference (**.CRF**) file on a data collector.

Cross-Reference File Discussion

Most database programs are capable of exporting a text file (***.TXT**) to be used by other applications. In many situations, a user needs to use the text file for a cross-reference file. A text file cross-reference file can be used by a LaserLite Mx with a memory card (refer to page 149). A DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx without a memory card cannot recognize a text file as a cross-reference file until it is converted into a ***.CRF** file. Cross-reference files created with Application Builder are automatically saved as ***.CRF** files.

The cross-reference text file must be comma or tab delimited. Do not place quotes around the records; however, quotes are optional around fields. Up to three description fields can be in the cross-reference file besides the key field. Fewer than three fields are acceptable, but do not use more than three fields. (If more than three fields are necessary, use **Vxcrf.exe** discussed on pages 184–187.) The text file should be in the following format:

Key<delimiter>**Field1**<delimiter>**Field2**<delimiter>**Field3**<cr/lf>

and saved as a text file with a **.TXT** extension. Note: If using tab delimiters, use a text editor that preserves tab characters (for example, Notepad or Wordpad, but not MS-DOS text editor).

You can convert the text file into a ***.CRF** file by either opening the text file in the Application Builder's **CRF** window, or by converting the text file with either the **Vxcrfw.exe** or the **Vxcrf.exe** programs located in the **Develop** folder.

To convert a text file into a ***.CRF** file with the Application Builder's **CRF** window, choose **Open** from the **File** menu; the **Open** window appears. At the bottom of the **Open** window is a **Files of type:** pop-up menu; choose **Any file** from the menu and open the text file. The text file is displayed in a **CRF** window. When you save the file, it is automatically converted and saved as a ***.CRF** file.

To convert a text file with the **Vxcrfw.exe** program, drag and drop the text file icon onto the **Vxcrfw.exe** program icon; a ***.CRF** file is automatically created. To convert a text file with the **Vxcrf.exe** program, run **Vxcrf.exe** from the command line, a ***.CRF** file is automatically created. See the following sections for complete information on using **Vxcrfw.exe** and **Vxcrf.exe**.

Vxcrfw.exe

The cross-reference file (***.CRF**) convert program for Windows is **Vxcrfw.exe**.

Vxcrfw.exe is a Windows executable program that converts an ASCII text file (***.TXT**) into a ***.CRF** file that can be used by a DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx without a memory card. The ASCII text file must be comma or tab delimited, have no quotes placed around records (quotes are optional around fields), no more than three description fields in addition to the key field, and saved as a text file.

To use **Vxcrfw.exe** to convert the text file into a ***.CRF** file, drag and drop the text file icon onto the **Vxcrfw.exe** program icon. A file with a **.CRF** extension is automatically created.

If the text file has 4 to 16 description fields in addition to the key field, you must use the DOS program **Vxcrf.exe** to convert the file. See the following page for information on using **Vxcrf.exe**.

Vxcrf.exe

The cross-reference file (***.CRF**) convert program for DOS is **Vxcrf.exe**. **Vxcrf.exe** converts an ASCII text file into a ***.CRF** file that can be used by a DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx without a memory card. The advantage to using **Vxcrf.exe** is that it can convert a cross-reference file with up to 16 description fields, in addition to the key field; whereas, the other convert programs are limited to three description fields. The ASCII text file must be comma or tab delimited, have no quotes placed around records (quotes are optional around fields), and no more than 16 description fields in addition to the key field. If more than three description fields are required, it is necessary to use the **-v** parameter in the command line. With three or fewer fields, the **-v** parameter is optional.

The syntax for the command line is:

vxcrf [-v<number of fields>] *filename.txt*

where *filename.txt* is the name of the text file and the **-v** parameter indicates the number of description fields (in addition to the key field) in the *filename.txt* file. **Vxcrf.exe** creates a file *filename.crf* that can be used as a cross-reference file by a DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx without a memory card.

The following steps describe how to:

- convert a text file (***.TXT**) into a ***.CRF** file,
- edit the application source code file to include the ***.CRF** file,
- compile the edited application source code file,
- and transfer the compiled application and ***.CRF** file to the data collector.

1. Create a comma-delimited or tab-delimited ASCII text file (***.TXT**) in the format:

   Key<delimiter>**Field1**<delimiter>**Field2**<delimiter>...**FieldN**<cr/lf>

   where **Key** is the data matched to the user input and **Field1** through **FieldN** are the description fields. **Field1** through **FieldN** are optional, but do not have more than 16 fields in addition to the key field. Note: If using tab delimiters, use a text editor that preserves tab characters (for example, Notepad or Wordpad, but not MS-DOS text editor).

2. To convert the text file into a ***.CRF** file, run **Vxcrf.exe** from the DOS command line:

   vxcrf [-v<number of fields>] *filename.txt*

   where the **-v** parameter indicates the number of description fields in the *filename.txt* file, in addition to the key field. If the number of description fields is three or less, the **-v** parameter is not necessary. The number of description fields cannot be greater than 16. This program will create a file *filename.crf*, which is a cross-reference file that can be used on a DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx without a memory card.

3. To use the ***.CRF** file in your application, open the application source code file (**.B** extension). Note: The application source code file is created when you choose **Export Binary** or **Export Source Only** from the Application Builder's **File** menu. See the **Export Binary** command section in the *Application Builder Manual* for information on creating the source code file.

4. In the application source code file, open the ***.CRF** file using the following syntax:

   OPEN *filename$* **FOR REFERENCE AS** #*filenumber%*

   The *filename$* is the name of the ***.CRF** file; it must be all uppercase and enclosed in quotes. The *filename$* must be the same name used in Step 2 previously (with the **.CRF** extension). The *filenumber%* must be a number between 0 and 31 that has not been used as the *filenumber%* in another **OPEN** statement.

It is recommended that the **ERR** function (which returns the most recent error condition) be called immediately after attempting to open a ***.CRF** file. For example:

```
open "FILE1.CRF" for reference as #1
 if err then
       gosub crf_error
 endif
```

See the *Videx BASIC Manual* for more information on the BASIC statements and functions.

5. In the application source code, you can find a record in the ***.CRF** file with a specified key field using the syntax:

   rec% = LOOKUP (*filenumber%, strexpr$*)

   where *strexpr$* is the key field and *filenumber%* is the number that was used in the **OPEN** statement to open this ***.CRF** file. Each record in a ***.CRF** file is assigned a unique number from 1 through *n* (where *n* is the number of records in the file). This unique number is returned from **LOOKUP**. The records are not necessarily stored sequentially, so do not assume that **rec%** corresponds to the row number in the ***.CRF** file. If the data string *strexpr$* is not found, **LOOKUP** returns 0.

6. Following a successful call to **LOOKUP** within the source code, the data can be retrieved from the ***.CRF** file using the syntax:

   fieldDesc% = LOOK$ (*fieldnum%*)

   where *fieldnum%* is the field value to be returned. If *fieldnum%* is 0, the key field itself is returned. If it is between 1 and 16, the corresponding data string from the ***.CRF** file is returned. If it is any other value, the results are unpredictable.

7. After making any changes to the application source code file, it must be compiled before it can be used on a data collector. To compile the application, use **Vxbasic.exe** using the following syntax:

   vxbasic *filename.b*

   where *filename.b* is the name of the application source code file (**.B** extension). The application is compiled and saved as an object file with a **.S** extension. (Note: If **Vxbasic.exe** encounters any errors in the application source code file during the compile process, it stops and displays the error and source code line on the screen.)

8. Transfer both the compiled application source code file (***.S**) and the ***.CRF** file to the DuraTrax, LaserLite, LaserLite Pro, or LaserLite Mx without a memory card using the syntax:

   DOWNLOAD [cmd.txt] [app.s] [REF.CRF] [sys.os] [-p*n*] [-d*n*] [-f1]

   where **cmd.txt** is the commands file (optional); **app.s** is the compiled application; **REF.CRF** is the ***.CRF** file and must be in all UPPERCASE letters; **sys.os** is the operating system (optional unless unit has been reset to monitor mode); **-p*n*** specifies the serial port (1–4; default is 1); **-d*n*** specifies the IR configuration (0 for Base Station or built-in IR transceiver, 1 for JetEye). The **-f1** parameter is only used if you want to transfer the application to the flash memory of the LaserLite Pro or LaserLite Mx.
   It is possible to transfer multiple ***.CRF** files using **Download.exe**; the filenames must be separated by spaces.

   The application must be loaded each time a ***.CRF** file is loaded, even if only the ***.CRF** file changes and the application itself has not. Both files must be included in the **Download.exe** command. See the **Download for DOS** section on page 10 for more information.

Cross-Reference File (*.CRF) Notes

If you have difficulty using a ***.CRF** file in the data collector, check the following:

- If using tab delimiters, create the text file for the ***.CRF** file using a text editor that preserves tab characters (for example, Wordpad or Notepad, but not MS-DOS text editor).

- Use **Vxcrf.exe** or **Vxcrfw.exe** to convert the text file into a ***.CRF** file.

- If the ***.CRF** file has more than three fields, in addition to the key field, use the **-v** parameter in the **Vxcrf.exe** command line.

- Both in the **OPEN** statement in the application source code file and in the **DOWNLOAD.exe** command string, refer to the ***.CRF** file using all uppercase letters.

- Transfer both the application and the ***.CRF** file into the data collector, even if only the ***.CRF** file has changed.

- If the ***.CRF** file will not transfer to the data collector, verify that the ***.CRF** file contains data and is not empty. The communications programs will not transfer an empty ***.CRF** file.

Chapter 8

Theory of Operation

This chapter contains information on:

- The operations of the DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx.

Theory of Operation Overview

The DuraTrax, LaserLite, LaserLite Pro, and LaserLite Mx operate on four AA alkaline or nickel-cadmium batteries. Input voltage can vary from 4 to 6 volts (V) under full load (90 mA). A linear regulator converts the battery voltage to +3.3 V. The processor is an 8051 type. Clock and RAM are backed up with a 3 V lithium battery. A boot loader (monitor program), residing in the processor ROM, handles program loading from the IR serial link.

All four units use similar CPU boards, but different scanner boards. The DuraTrax has a contact scanner and the LaserLite, LaserLite Pro, and LaserLite Mx have a laser scanner. Each unit has a nose board in the head of the device for the LEDs, lithium battery, and beeper.

Some of the I/O functions available for applications are: Real-time clock (RTC), analog-to-digital converter (ADC) for battery, Touch Memory button contact, contact bar code reading, LEDs, beeper, serial communications, display, keyboard, and power management.

Power Up Sequence

With batteries installed (greater than 4.0 V) and lock switch ON, a wake-up from a keypress or Touch Memory button contact resets the CPU. The processor initiates a power-on sequence and then executes code from the XRAM (external RAM). The program sets up all latches as required. A search is carried out to determine the required action: button contact, keypress (scan or scroll), or serial communications. The LCD is activated to display human readable characters.

Battery Monitoring

Battery conditions can be monitored by the ADC so that a low-battery alert is implemented by the software. For alkaline batteries this would be 4.5 to 4.0 V. Different battery chemistries (such as, alkaline or NiCad) may affect the threshold chosen. Both the default application and the application templates begin issuing low voltage warnings at 4.8 volts. More aggressive warning messages are triggered at 4.5 volts. This provides a 0.9 V margin above the operation cut-off for batteries that have greater than normal internal resistance.

A power-fail indicator monitors the battery source. It sends an interrupt to the CPU if the source falls below 3.6 V. At this point, the scanner circuits may experience reduced reading performance. If required, a CPU/RAM cleanup can be completed before shutting down. The CPU should always be prepared for the eventual loss of power from battery removal before entering STOP mode. A power failure will not wake up a sleeping CPU.

A hardware-reset circuit monitors the low-battery alert (VCC). Should the battery voltage fall below where the CPU and RAM operate, at about 3 V, the CPU is placed in reset and the RAM Chip Select line is deactivated.

Power Down Sequence

Operating system software places all devices in low-power modes before entering STOP mode of the CPU. These include scanner, LCD, IR, and LEDs.

Boot Loader (Monitor Program)

Boot loader residing in processor ROM allows update of the RAM from the IR port. See Chapter 2 for more information on the monitor program.

Communications

Communications for transferring software and data is through the IR port.

Memory Access

(Refer to the System Memory Map Diagrams on pages 48–50.)

RAM: 128K x 8 bit (LaserLite and DuraTrax)
256K x 8 bit (LaserLite Pro and LaserLite Mx)

MEMORY MAP

Data and program space not segregated.

A15 = Low	A0–A14	Access base 32K block.
A15 = High	A0–A14	AD15, AD16, AD17 select one of 32K blocks.

If both IOSEL1 and A15 High, A4 and A5 select I/O device to Read or Write. Note: If both IOSEL1 and A15 High, No DATA or PROG access to RAM, reserved for I/O space only.
To access latch (LEDs, beeper, etc.), keyboard, LCD.
Program execution from the base 32 KB.

Times

(Note: The times are approximate values.)

Battery removal VCC holdup time (Time from power fail to reset)	1 mS
Reset duration from sleep mode initiated by key activity	40 mS
Power cycling reset	250 mS
RAM access time maximum	100 nS

Currents

(Note: The currents are approximate values.)

CPU Stop Mode with all devices OFF	800 µA
CPU Idle Mode with LCD ON	8 mA
CPU Run Mode with Scanner ON	80 mA

Voltages

(Note: The voltages are approximate values.)

ADC battery range	3.6 V to 6.5 V
Device operation with scanners	4.0 V
Battery power fail threshold	3.6 V
Battery reset level	3.3 V

Environmental Testing

CE mark

FCC

Definition of Terms

ADC	Analog to Digital Converter
CPU	Central Processing Unit
IRAM	CPU Internal RAM (Random-access memory)
IrDA	Infrared Communications
IR	Infrared
LCD	Liquid Crystal Display
RAM	Random-access memory
ROM	CPU Boot ROM (Read-only memory)
RTC	Real-Time Clock
XRAM	External 128K RAM

Circuit Description

Main Board

The main board contains the CPU/RAM and controls the activity of the data collector.

Battery supply - The four AA batteries supply the power to the data collector. A diode protects the device from reverse polarity. A regulator supplies the 3.3 V to the CPU, RAM and logic gates. A second diode along with its regulator supply 3.3 V to the IR diode, LCD doubler, and scanner. Another regulator monitors the battery voltage should it drop below 3.6 V and sends an interrupt to the CPU. The LCD requires 5 V from a doubler and is regulated to 5 V. The power to the scanner is controlled by a transistor which is switched by the output latch.

Wake-up circuit - The wake-up circuit resets the CPU from stop mode. Any activity from the keys, button contact, or alarm will initiate the reset. The switch disables wake up when switched OFF, but does not disable RTC wake-up process. Reset time is approximately 30–50 ms to give the IR crystal time to restart.

Button touch - The button touch circuit toggles and senses the touch line as needed to communicate with Dallas Semiconductor Touch Memory buttons (also known as iButtons).

Real-time clock (RTC) - The RTC performs many functions. The Watch Dog function, when enabled, monitors the CPU for activity. The ADC is used to monitor battery voltage. The Reset circuit monitors the VCC line for a safe operating range. A non-volatile RAM controller monitors the VCC line for placing the RAM in low-power mode. The lithium battery is also switched through the RTC IC, and of course the real-time clock and alarm. A 32 kHz crystal maintains the time of day.

CPU - 8051 family of 8-bit processor. The CPU, under software control, handles all of the needed operations to communicate with other devices. An 11.059 MHz crystal controls the timing.

Lithium battery - A 3 V lithium battery is located on the nose board. The battery backs up the RAM and RTC should the AA batteries be removed. The lithium battery is recharged through a 3.3 V regulator and diode at a constant voltage and current limit. The RTC controls the switch over, when VCC drops below 2.5 V.

IR communications - A controller controls the detection and modulation of the IR diodes for communications to a compatible IrDA docking device. A 3.6 MHz crystal controls the timing. The IR_TX and IR_RX lines are shared with the RTC. The IR IC is placed in float mode and low-power mode when not in use.

RAM	(DuraTrax and LaserLite) 128K x 8 bits RAM IC. Backed-up by the lithium battery. (LaserLite Pro and LaserLite Mx) 2-128K x 8 bits RAM IC. Backed-up by the lithium battery.
Digital Logic	The following logic devices are used on the CPU board: **Address Latch** Multiplexes address lines A0–A7. **I/O Select** A decoder multiplexes I/O select lines. **Buffer** Input buffer for keys and output buffer for LCD. **Latch** A latch IC outputs control lines for the devices. **LCD** The LCD is 2 x 16 character display.

DuraTrax Scan Board

The scan board modulates the optic read head and demodulates the AM signal to a digitized waveform to be processed by the CPU board.

The read head is a self-contained LED and detector diode focused on the bar code.

A low-pass filter filters the modulated signal and demodulates the AM signal.

A modulator and transistors modulate the read head diode at about 50 kHz.

A peak detector detects the average threshold level and squares the output for digital processing.

A regulator regulates the battery source to 3.3 V.

LaserLite, LaserLite Pro, and LaserLite Mx Scan Board

The scan board translates the laser 5 V operation to the 3.3 V logic of the CPU board.

The translators translate 3.3 V logic to 5 V logic of laser.

A regulator regulates and limits the battery voltage to 5 V maximum. When battery voltage drops below 5 V, the laser voltage follows down to 3.3 V operation.

Base Stations

The LaserLite DuraTrax Base Station is designed to accept DuraTrax and LaserLite data collectors. Each LaserLite DuraTrax Base Station can hold up to two units.

The LaserLite Pro Base Station is designed to accept LaserLite Pro and LaserLite Mx data collectors. Each LaserLite Pro Base Station can hold one unit.

Videx software handles the transfer process. A computer's RS-232 serial port is the intended communicating device. An expansion port is supplied on each device that allows multiple Base Stations to be connected to a single serial port.

Notes:

Appendix A

Modem Communications Hook-up Configurations and Cable Pin Outs

Contents

Note on Signal Direction Convention:

RS-232 signal wires are given names that stay with the same wire as it goes between the two devices being connected. Signals that imply a direction, such as “Receive Data,” are named from the perspective of the “Terminal” (DTE) device and may therefore appear to be backward in terms of signal direction when applied to the “Modem” (DCE) device on the other end of the cable. In the lists of pin assignments in this section, an indication of signal direction from the point of view of the device to which the connector is attached has been included in addition to the signal name.

The following illustration shows you how to connect the Base Station to a modem for remote data transfer.

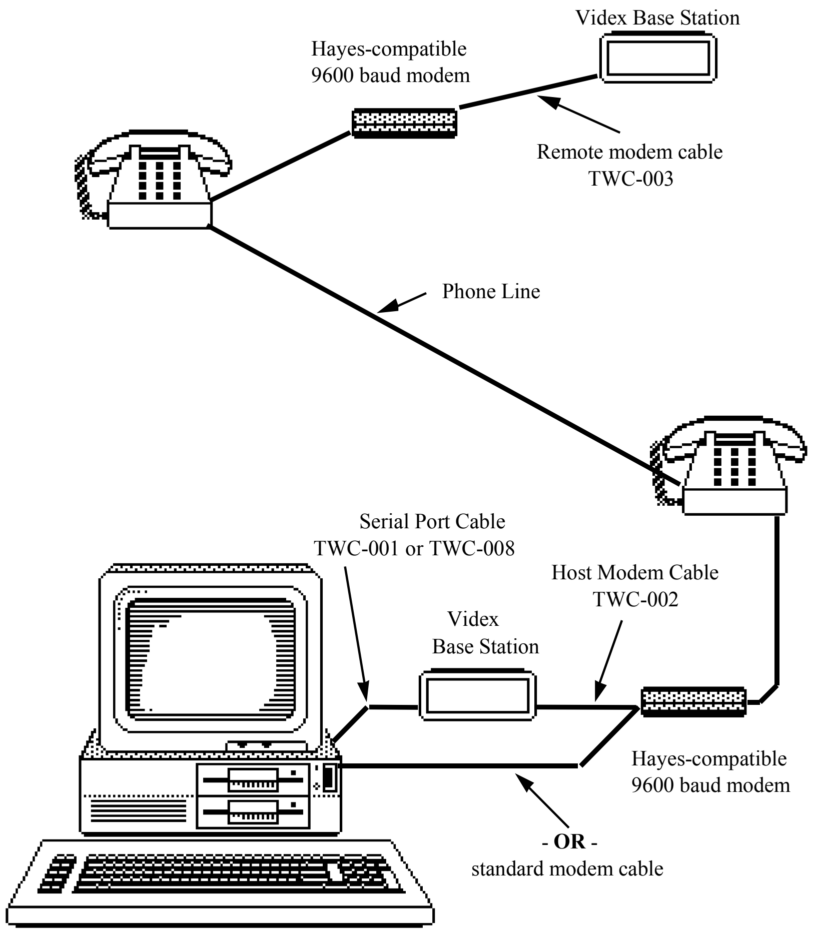


Figure A-1 Modem Transfer Set Up

TWC-002 Cable — Host Modem to Base Station

The cable used to connect a modem to Base Station at the host modem location is shown in Figure A-2.

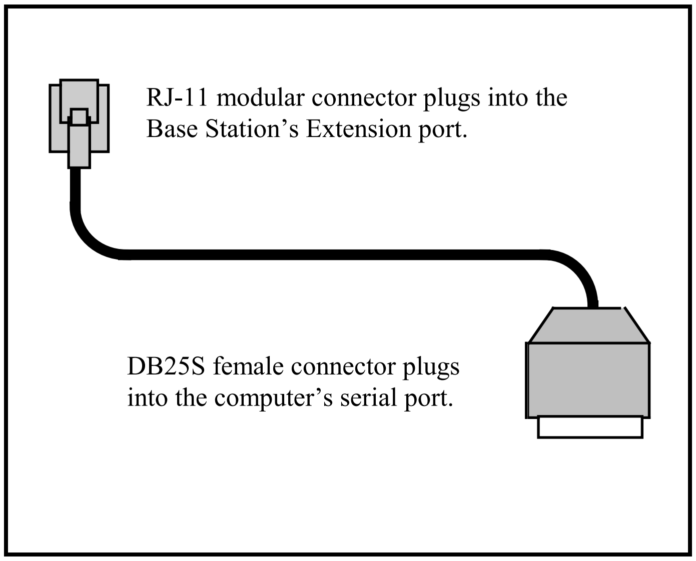

RJ-11 modular connector plugs into the
Base Station's Extension port.
DB25S female connector plugs
into the computer's serial port.

Figure A-2 Host Modem Cable

Figure A-3 shows the pin configuration of the DB25P connector in relation to the RJ-11 modular connector.

Host Modem to Base Station Cable

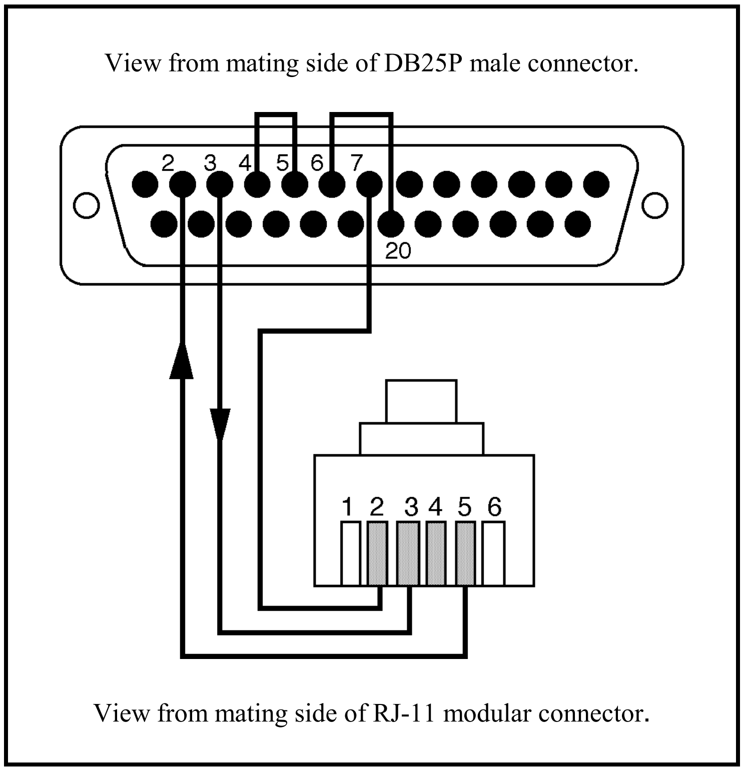

View from mating side of DB25P male connector.
2
3
4
5
6
7
20
1 2 3 4 5 6
View from mating side of RJ-11 modular connector.

Figure A-3 Host Modem Cable Configuration

25-Pin Assignment		**Modular Connector Pin Assignment**	
2	TXD Transmit Data (In)		
3	RXD Receive Data (Out)	2	RXD (In)
5	CTS Clear to Send (Out)	4	TXD (Out)
6	DSR Dataset Ready (Out)	5	Ground
7	Ground		
20	DTR Data Terminal Ready (In)		

TWC-003 Cable — Remote Modem to Base Station

The cable used to connect a modem to the Base Station at the remote modem location is shown in Figure A-4.

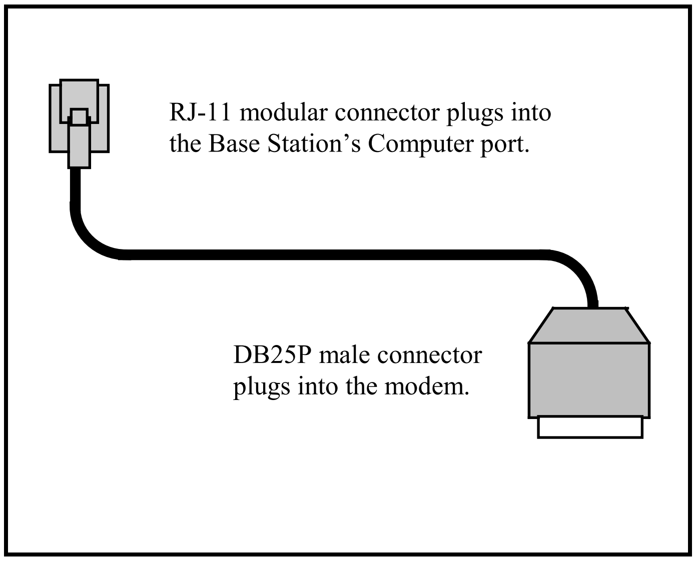

RJ-11 modular connector plugs into the Base Station's Computer port.
DB25P male connector plugs into the modem.

Figure A-4 Remote Modem Cable

Figure A-5 shows the pin configuration for the DB25P connector in relation to the RJ-11 modular connector.

Remote Modem to Base Station Cable

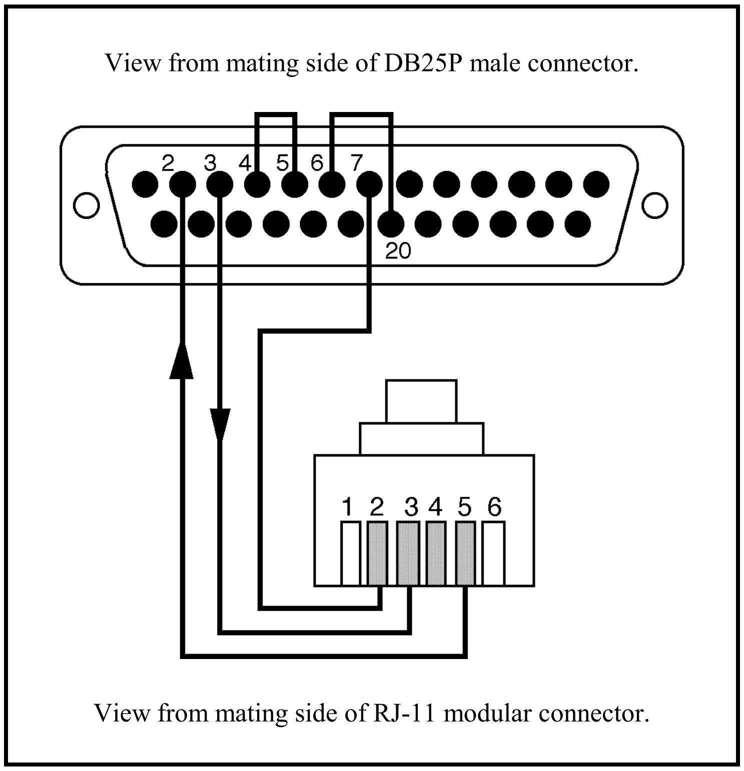

View from mating side of DB25P male connector.
2
3
4
5
6
7
20
1 2 3 4 5 6
View from mating side of RJ-11 modular connector.

Figure A-5 Remote Modem Cable Configuration

25-Pin Assignment		**Modular Connector Pin Assignment**	
2	TXD Transmit Data (In)		
3	RXD Receive Data (Out)	2	Ground
4	RTS Request to Send (In)	3	RXD (In)
5	CTS Clear to Send (Out)	5	TXD (Out)
6	DSR Dataset Ready (Out)		
7	Ground		
20	DTR Data Terminal Ready (In)		

Appendix B

Low-Level LaserLite Mx Memory Card Formatting Sequences

Contents

Formatting an Unformatted Memory Card

When formatting a memory card for a LaserLite Mx that is currently unformatted or does not have a recognized format, the DMS software must be explicitly loaded and invoked before actual card formatting can begin. The necessary steps (in order) are as follows:

1. Place the LaserLite Mx's lock switch in the locked position (towards the lock icon).

2. Insert the memory card into the LaserLite Mx memory card slot.

3. Flip the lock switch back to the unlocked position (away from the lock icon). The system should boot in firmware mode.

4. Issue the following command to unlock the physical writing functions:

   N &0129

5. Upload the Data Management System (DMS) software (plus a two-byte CRC for the entire application) into XRAM, starting at memory address 0000. This is done by repeated use of the command:

   X {address} {byte-sequence}

   where **address** is the physical address in memory and **byte-sequence** is the portion of the DMS being uploaded. The maximum size of the DMS program (including the CRC) cannot exceed 24K bytes. This means the maximum address is 5FFF.

6. Once the DMS software is uploaded, transfer control to it from the firmware by issuing the command: **G**

7. Issue the command: **N &1092**
   again to begin card initialization. This step takes approximately 5 to 10 minutes to complete.

8. When the card is initialized, assign an ID name to the card with the open (**O**) command:

   O D "nnn…"

   where "**nnn…**" is the card ID. The ID can be up to 18 characters in length and can contain printable character.

9. Open a card boot file by issuing the command:

   O B "CRD"

   and load the CRC-terminated boot file using repeated calls to the **A** command.

10. Open a CPU boot file by issuing the command:

    O B "CPU"

    and load the CRC-terminated boot file using repeated calls to the **A** command.

At this point, the card formatting process is complete. The card is now ready to receive and store application and/or data files.

Formatting Previously Formatted Memory Cards

When formatting a memory card that was previously formatted, the process is much less involved. The necessary steps are as follows:

1. If the entire card contents (including the card boot file and CPU boot file) are to be replaced, erase all existing data on the card by issuing the command:

   N &1092

2. If only the one of either the card boot file or CPU boot file need to be upgraded, the file to be upgraded may be individually erased by issuing one of the following pair of commands:

   D B "CRD"
   K &1092

   or

   D B "CPU"
   K &1092

3. If the card boot file has been erased, open a card boot file by issuing the command:

   O B "CRD"

   and load the CRC-terminated boot file using repeated calls to the **A** command.

4. If the CPU boot file has been erased, open a CPU boot file by issuing the command:

   O B "CPU"

   and load the CRC-terminated boot file using repeated calls to the **A** command.

The card formatting process is complete. The card is now ready to receive and store application and/or data files.

Appendix C

Error Codes

Contents

Communication Errors

Communication Errors Returned by Vxcom:

Xmodem Errors (LaserLite, DuraTrax, LaserLite Pro, & LaserLite Mx)

Vxcom Error #	Description
16001	Gave up waiting for a character.
16002	Aborted transmission.
16003	Got out of sync with other party in transmission.
16004	Keep missing parts of packets; there may be too much noise on line.
16005	Received the start of the packet, but rest of packet is badly formed.
16006	Received a well formed packet, but the check digits were wrong. May have noise on the line.

General Communication Errors (LaserLite, DuraTrax, LaserLite Pro, & LaserLite Mx)

Vxcom Error #	Description
18000	Cannot open the serial port.
18001	Unknown command in the commands file.
18002	Error in communications.
18003	**C** and **I** commands in commands text file require identifiers.
18004	No such serial port.
18005	Error accessing a file.
18006	Not enough memory to continue.
18007	Error talking to the serial port.
18008	One of the command line arguments in the commands file does not make sense.
18009	Loops in the commands file are nested too deeply.
18010	The file size exceeds the device's capacity.
18011	There is no application file to load with the **OS**.
18012	You must specify either an **OS** file or an image file to load in flash.
18013	Image files (***.IMG**) can only be loaded into flash.
18014	The structure of a cross-reference file is corrupted.

General Communication Errors (LaserLite Mx)

Vxcom Error #	Memory Module Error #	Description
21001	01	Unrecognized memory card. This version recognizes Toshiba's SSFDC 2, 4, and 8 MB memory cards.
21002	02	Unrecognized memory card.
21003	03	Syntax error.
21004	04	CRC of command did not match.
21005	05	Unknown command.
21006	06	Missing parameters for this command.
21007	07	Incorrect parameters for this command.
21008	08	Binary file.
21009	09	Incorrect file type.
21010	10	Too much data in the command.
21022	22	No ID file.
21023	23	No data.
21031	31	No such file exists.
21032	32	No file opened.
21033	33	Too many files (>60 files).
21034	34	The page has already been written to 4 times. (Note: Toshiba only allows writing to a page 4 times.)
21035	35	No such record exists.
21036	36	End of file.
21037	37	Beginning of file.
21038	38	Timeout occurred.
21039	39	Memory full.
21040	40	Write/protect encountered.
21041	41	Record too large (>1024 bytes).
21042	42	Field too large (>255 bytes).
21043	43	Some of the physical functions are locked to avoid data corruption. Try: **N &0129** to unlock.

Vxcom Error #	Memory Module Error #	Description
21051	51	File management error (control data was changed).
21052	52	Sequential file corrupted (accidental power failure).
21053	53	Data corrupted.
21054	54	CRC of boot file did not match (program data may be corrupted).
21061	61	Memory card program failed (card may be worn out).
21062	62	Block erase failed (card may be worn out).
21063	63	Unknown memory card format.
21064, 21065	64, 65	First block of memory is bad (broken card).
21066	66	Too many bad blocks (card may be damaged).
21067	67	Most of the reserved control blocks are bad (card worn out).
21068	68	The memory card has been erased more than 32,768 times (each **K &1092** or **N &1092** counts as one erasing).

Memory Card Errors

Error Codes for LaserLite Mx Only

Global Errors Returned by Operating System via CARDCMD:
These errors prevent sending a statement to the memory card software system (LaserLite Mx only).

Error	Description	Possible Cause
-1	Communications with the memory card timed out.	The memory card software system is busy processing the last command set it received or it didn't have enough time to finish processing the current command. In either case, try using the **CARDCMD** statement with the [!] option on commands that require more time than 800ms to process.
-2	A memory card module was not detected at startup.	**CARDCMD** reads the system startup flags before attempting to communicate with the memory card software system.
-3	A memory card was not detected at startup.	If a memory card is inserted, the system should be restarted to properly set the flags that a card is present.
-4	The memory card does not have a recognized format.	Unformatted card.
-11	The memory card processor is asleep and cannot receive a command.	Removing the memory card without first turning off the LaserLite Mx can cause this error.

Errors Returned by Memory Card Software System via CARDSTATUS (LaserLite Mx only):

Error	Description	Note
-1	Communications with the memory card timed out.	The memory card software system does not respond.
-2	A memory card module was not detected at startup.	As with the **CARDCMD** statement, **CARDSTATUS** first checks the system flags before attempting to communicate with the memory card module. Errors -2 and -3 will be returned if the system flags indicate.
-3	A memory card was not detected in the module at startup.	
-5	The CRC of the return from the memory card system does not match the one calculated by **CARDSTATUS**.	System communications problem between the main CPU and the memory card module.
-6	The return from the memory card is unrecognized.	Same as above.
-7	The designated string is too short for return from the memory card.	Dimension the string memory variable that receives data from **CARDSTATUS** to a larger value.
-8	No **CARDCMD** was issued since the last call to **CARDSTATUS**.	This indicates a coding error. The system expects **CARDCMD** and **CARDSTATUS** to be called in pairs.
-10	Card is still busy when **CARDSTATUS** called.	The memory card software system is still processing an instruction.
-11	Card is asleep when **CARDSTATUS** called.	Removing the memory card during operations can cause this error.
-15	The return from the memory card has exceeded the 2K buffer.	This should occur only with 1K data records containing '&' as every 2nd character.

Index