EPSON
EXCEED YOUR VISION

S1D13513

PLAY Diagnostic Utility

Document Number: X78B-B-001-01.02

SEIKO EPSON CORPORATION

Rev.1.02

Table of Contents

1. Introduction

PLAY is a diagnostic utility which permits a user to read from and write to all the registers and memory of the S1D13513. Commands are received from the standard input device, and messages are sent to the standard output device.

Commands may be entered interactively by a user, or can be executed from a script file. Scripting provides a powerful interpretive language that allows command sequences to be used repeatedly without re-entry.

Note

This program is a utility to assist with testing and evaluating the S1D13513. It is not a display driver for any operating system.

This user manual is updated as appropriate. Please check for the latest revision of this document at vdc.epson.com before beginning any development.

We appreciate your comments on our documentation. Please contact us via email at vdc-documentation@ea.epson.com.

1.1. PLAY Supported Evaluation Platforms

PLAY is available as an executable for PCs running Windows® XP/Vista.

Note

In order to run PLAY on PCs running Windows, the device driver **S1D13xxxUSB.sys** must be installed. For further information on this device driver, refer to the documentation contained within the driver package.

The latest version of both the executable and source code is available on the Epson Research and Development website at vdc.epson.com.

2. Installation and Usage

To install the PLAY program, copy the file **play.exe** to a directory (e.g. C:\ S1D13513).

To start the PLAY program from the Windows GUI, navigate to the directory where the program was installed and double click the PLAY.EXE icon.

To start the PLAY program from a command prompt, open a command prompt window, navigate to the directory where PLAY was installed and type:

play [command] [/?] [/nocard]

Where:

/?	Displays copyright and program version information
/nocard	Allows PLAY to be run without a S1D13513 evaluation board
command	Play commands entered from the command line.

3. Play Commands

The PLAY command interface is designed to permit fast, easy access to the majority of features supported by PLAY. There are three ways to enter commands into PLAY.

- Interactive
  Interactive commands are typed at the PLAY program prompt. The PLAY prompt is the = (equal) sign.

- Scripted
  Scripting permits playback of command sequences and supports simple script processing. Scripting is a powerful feature of PLAY and is discussed in detail in Section 5. Scripts are stored as an ASCII text file.

- Startup command line
  Short commands may be entered on the command line. Command line instructions with multiple arguments must enclose the arguments in double quotes.
  (e.g. play “f 0 14000 AB” q).

When required, multiple commands can be run from one entry by separating each command with a semicolon. For example, the following entry will display the contents of the registers at index 0, index 2, and index 4.

```
x 0; x 2; x 4
```

Help on specific commands is available at the play prompt by typing the name of the command followed by a “?” character without any other arguments. The HELP command provides a summary of the available commands and information on the tokens and features (bitfields) that are associated with each command.

The following keystrokes can be used at the PLAY command prompt:

ESC	Clears the command line
CRTL-R	Recalls the last command
CTRL-Backspace	Deletes the last word
Up arrow, down arrow	Scrolls through the command history

3.1. Basic Play Commands

PLAY includes several basic commands which are designed to give the user direct control over the registers and memory of the S1D13513. The commands in this group are summarized in the following table.

Command	Description
D[8|16|32] [StartAddr [EndAddr|Len]]	**Dump the contents of memory to the display**
F[8|16|32] StartAddr EndAddr|Len Data1 [Data2 Data3 ...]	**Fill memory with the specified data**
I[R][16] [Addr [Count|Data…]]	**Performs writes to the I2C device**
IAP[R] [Addr [Data…]]	**Performs a write to the Agilent camera**
IAS[R] [Addr [Data...]]	**Performs a read from the Agilent camera**
ID [DeviceID]	**Sets the device ID of the I2C slave**
IFM [0|1]	**Sets or clears I2C Fast mode.**
II [DeviceID]	**Initializes the specified I2C slave**
INIT [NOflags|ONLYflags]	**Initializes the LCD Controller**
IRS [0|1]	**Sets or clears I2C repeated start condition for reads**
IT	**Tests all slave IDs to find devices**
M	**Show current Mode information**
MM	**Show Memory Map (memory usage layout)**
Q	**Quit the program**
R[8|16|32] [Addr1 Addr2 Addr3 ...]	**Read data from memory**
S[8|16|32] StartAddr EndAddr|Len Data1 [Data2 Data3 ...]	**Search memory for the specified data**
SET [Token [Expression]]	**Set values for registers, features, and variables**
SHOW [MAIN|PIP1|PIP2|MEM:addr] [FILL|GRID] [H:n] [W:n] […]	**Shows test patterns on the display**
W[8|16|32] [StartAddr [Data1 Data2 Data3 ...]]	**Write the specified data to memory**
X[A|B] [8|16|32] [Index [Data]]	**Read/Write one controller register**
XF[B][8|16|32] Index Cycles Data1 [Data2 Data3 ...]	**Register Fill**
XR[B][8|16|32] Index Cycles	**Register Read**
?|HELP [RegIndex|SearchText]	**Provides help on a specified register or search parameter**

D[8|16|32] [StartAddr [EndAddr|Length]]
Dumps (displays) the contents of memory from the specified address range. The Dump command differs from the Read command in that it is intended to read **blocks** of memory. If the controller supports burst reads, this feature is used for reading the memory.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last D command. If no previous D command has been issued, the unit size defaults to 8-bit.
StartAddr	The start address in memory to read data from. If StartAddr is not specified, the memory dump will begin where the previous Dump ended. If a period "." is specified for the start address, the dump is done from the same start address as used in the last D command.
EndAddr|Len	The end address of the memory range to dump. The end address can be specified as a physical memory address or as the length (in terms of unit size) of the memory dump. If a length is specified, it must be prefixed with "L". For example, specifying L8 when the unit size is 16-bit causes the D command to dump 8 words from the starting address. If an "L" is not specified, PLAY assumes that the value specified is a physical memory address. When this argument is omitted the previous Dump length is used. The default, if the length has not been changed is 256 bytes.

F[8|16|32] StartAddr EndAddr|Len Data1 [Data2 Data3 ...]
Fills the specified address range in memory with the given data. If the range to fill is larger than the number of data elements, the data elements are repeated as many times as necessary.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last F command. If no previous F command has been issued, the unit size defaults to 16-bit.
StartAddr	The start address in memory to write data to. If a period "." is specified for the start address, the fill is done from the same start address as used in the last F command.

EndAddr|Len	The end address in memory to stop filling with data. The end address can be specified as a physical memory address or as the length (in terms of unit size) of the memory fill. If a length is specified, it must be prefixed with "L". For example, specifying L8 when the unit size is 16-bit causes the F command to fill memory starting from the start address with 8 words. If an "L" is not specified, PLAY assumes that the value specified is a physical memory address. EndAddr must be specified.
data#	The data value(s) to fill memory with. If multiple data values are given, the pattern is repeated through memory until the specified end address or length is reached. Data values can be specified as text, numbers, or combinations of both. Numbers are assumed to be hexadecimal values unless otherwise noted with the correct suffix (binary = i, octal = o, decimal = t, hexadecimal = h). For example, 101i = 101 binary. To specify text values, enclose text data inside double quotes.

I[R][16] [Addr [Count|Data…]]
Reads or writes the current I2C device. The data transfer can be in raw or processed mode. In raw mode there is no interpretation of the data while in processed mode Addr specifies an offset address where data transfers will begin in the I2C device.

Where:

R	Raw mode. When Raw mode is selected there is no data interpretation performed.
16	Selects 16-bit transfers. If this parameter is not specified, an 8-bit transfer is performed. Not all devices support 16-bit data transfers.
Addr	The device sub-address. Addr is not used for Raw mode transfers.
Count	The number of reads to perform. When there is no data given, the I command reads Count data cycles from the I2C device.
Data	Data value(s) to be written to the device.

IAP[R] [Addr [Data]]
IAS[R] [Addr [Data]]
Commands to support accessing an Agilent camera processor and sensor. Typically, accessing the control stacks on these cameras is a multi-step process. The Agilent specific commands simplify the process by indexing into the processor or sensor stacks.
IAP is used for accessing the Agilent camera processor.
IAS is used for accessing the Agilent camera sensor.

Where:

R	Read from the processor or sensor. When R is not specified, the data is written to the specified address within the processor or sensor.
Addr	The offset within the processor or sensor to access.
Data	Data value(s) to be written to the Addr offset.

ID [DeviceID]
The I2C subsystem only deals with one device at a time. The ID command is used to set or determine the current active device ID.

Where:

DeviceID	The device ID of the I2C slave.

IFM [0|1]
Clears or sets I2C fast mode operation (100KHz/400KHz). When a 0 is specified, the slower 100KHz I2C clock is selected. If a 1 is specified, the faster 400KHz I2C clock is selected. When the state is not specified the current fast mode state is returned.

Where:

0	Clears I2C Fast mode.
1	Sets I2C Fast mode.

II [DeviceID]
Initializes I2C devices that the HAL is aware of (typically used for cameras). If DeviceID is not specified, all configured I2C devices are initialized. If DeviceID is specified, then only the specified device is initialized.

Where:

DeviceID	The device ID of the I2C slave to be initialized.

INIT [NOflag|ONLYflag|…]
Initializes the LCD controller by setting the registers, programmable panels, and display memory to pre-determined configuration values. When no parameters are defined the initialization is performed based on the settings from the 13513 CFG program. Parameters can be used to override and fine tune the initialization sequence. The override parameters are:

Where:

NORESET	Do not issue a software reset.
NOI2C	Do not initialize the I2C interface.
NOREGS	Do not initialize controller registers
NOLCDS	Do not initialize the panel
NOLUT	Do not initialize the Look-up Table (LUT)
NOVRAM	Do not clear display memory
ONLYRESET	Only issue a software reset
ONLYI2C	Only initialize the I2C interface
ONLYREGS	Only initialize the registers
ONLYLCDS	Only initialize the panel
ONLYLUT	Only initialize the Look-up Table (LUT)
ONLYVRAM	Only clear the display memory

IRS [0|1]
Clears or sets I2C repeated start. Specifying 0 clears the repeated start state, while specifying 1 sets the repeated start state. If the state is not specified, then the current repeated start state is returned.

IT
Tests all I2C slave IDs to determine which, if any, devices exist. This command is primarily used for debugging purposes.

M
Show the current Mode information. This includes information about the active display interface, dimensions and details of configured windows, and clock frequencies. An example of the information that may be displayed for the M command is shown below.

```
=m

 DEVICES   [*ON]     WxHxBPP DESCRIPTION
 --------------- ----------- ---------------------------------
*Display          640x480x08 General TFT/ND-TFD
*Camera 1       1280x1024xNA Omnivision OV9650 SXGA 1280x1024
 Camera 2         352x288xNA Agilent ADCM-1650 CIF 352x288

 DIMENSIONS       WxHxBPP   STRIDE START  SADDR
 ------------   ----------- ------ ------ ----------------------
*Main Window     640x480x08    640 N/A    00000000h + bias of 0h
 PIP1 Window     320x240x08    320 0,0    0004B000h + bias of 0h
 PIP2 Window         2x2x08      2 0,0    00096000h + bias of 0h
*View Resizer    320x240xNA    320 0,0
 Capture Resizer 640x480xNA    640 0,0

 CLOCKS     FREQ        SOURCE
 ---------- ----------- --------------------------------------
     INCLK1  10.000 MHz OSC1CLK
     INCLK2  10.000 MHz OSC1CLK
 PLL1REFCLK  10.000 MHz INCLK1/1
 PLL2REFCLK  10.000 MHz PLL2 Divided Source/1
   SDRAMCLK 100.000 MHz PLL1REFCLK*(9+1) [PLL1 USED]
  LCDSRCCLK  40.000 MHz PLL2REFCLK*(3+1) [PLL2 USED]
     SYSCLK  50.000 MHz SDRAMCLK/2
     KEYCLK  50.000 MHz SYSCLK/1
  PWMSRCCLK  50.000 KHz SYSCLK/1
    I2CCLK    4.545 MHz SYSCLK/11
  CM1CLKOUT  16.666 MHz SYSCLK/3
  CM2CLKOUT  50.000 MHz SYSCLK/1
    LCDDCLK  20.000 MHz LCDSRCCLK/2
    LCDSCLK  40.000 MHz LCDSRCCLK/1
```

MM
Shows the memory map layout for the S1D13513. The memory map consists of an address or address range followed by the memory usage. An example of the information that is displayed for the MM command is shown below.

Memory Map information is derived by reading and interpreting the registers associated with the various controller features.

```
=mm
008F1000h          -Back Buffer Start Address
008A6000h          -PIP2
0085B000h          -PIP1
00810000h          -Main Image
00810000h          -YRC Write Start Address 1
00810000h          -YRC Write Start Address 0
00810000h          -YRC Write Start Address 2
00810000h          -Sprite Frame Buffer 0 Start Address
00810000h          -Sprite Frame Buffer 1 Start Address
00810000h          Display Memory
00610000h-00A0FFFFh Registers
```

```
00610000h          Chip Base Address
```

Q
Quits the PLAY program.

R[8|16|32] [Addr1 Addr2 Addr3 ...]
Reads from the specified memory address location(s) and displays the data values on the screen. Unlike the Dump command, the Read command reads only from the memory locations specified.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last R command. If no previous R command has been issued, the unit size defaults to 16-bit.
Addr#	The address location(s) to read data from. If Addr is not specified the Read address will be the last specified address. Multiple addresses can be specified. Each address will be read and the memory contents displayed.

S[8|16|32] StartAddr EndAddr|Len Data1 [Data2 Data3 ...]
Searches a specified memory range for a given data pattern and displays the address(es) of where the data pattern matches the search pattern.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last S command. If no previous S command has been issued, the unit size defaults to 8-bit.
StartAddr	The start address in memory to start searching for the given data from. If a period "." is specified for the start address, the search is done from the same start address as used in the last S command.
EndAddr|Len	The end address in memory to stop searching for the given data at. The end address can be specified as a physical memory address or as the length (in terms of unit size) of the memory search. If a length is specified, it must be prefixed with "L". For example, specifying L8 when the unit size is 16-bit causes the S command to search memory starting from the start address for 8 words. If an "L" is not specified, PLAY assumes that the value specified is a physical memory address. If a period "." is specified for the end address or length, the search is done until the end

address or length used in the last S command is reached. An EndAddr or a Len must be specified.

Data#	The data value(s) to search memory for. When multiple data values are given, the specific sequence is searched for. Data values can be specified as text, numbers, or combinations of both. Numbers are assumed to be hexadecimal values unless otherwise indicated with the correct suffix (binary = i, octal = o, decimal = t, hexadecimal = h). For example, 101i = 101 binary. To specify text values, enclose text data inside double quotes.

SET [Token [Expression]]
Sets or gets the values for S1D13513 registers, features (bitfields), local variables, and global variables. If an expression is specified, the token is set to the value of the expression. If no expression is specified, the value of the token is displayed. If no token is specified, the SET command lists all user defined variables.

Where:

Token	The register index, feature (bitfield), local variable (i.e. $var), or global variable ($$var) that is to be set or displayed. If the specified token is a variable that does not exist yet, it is created.
Expression	The value to set the specified token to. If a period "." is specified for the expression, the user will be prompted for input. Numbers are assumed to be hexadecimal values unless otherwise specified with the correct suffix (binary = i, octal = o, decimal = t, hexadecimal = h). For example, 101i = 101 binary.

Note

At the prompt, return values from a function or script can be assigned to a token using the following construct.

token = fcnExpression

Where:

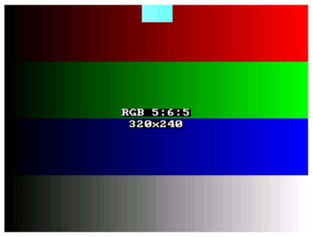
token	The register, feature (bitfield), or variable being assigned to.
fcnExpression	The function or script call that the return value is being accepted from.

Examples:

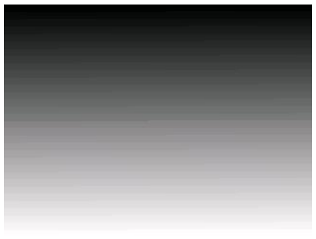
set 28 14	Sets REG[28] to 14 (hex)
set 28	Prompts the user for a value for REG[28]
set reg[28] 3	Sets REG[28] to 35 (hex)
set hdp 320t	Sets the HDP feature to 320 (dec)
set clki "27.000 MHz"	Sets the CLKI feature to frequency “27.000 MHz”
set $MyVar 42	Sets and/or creates a local variable to 42 (hex)
set $$Foo 38t	Sets and/or creates a global variable to 38 (dec)

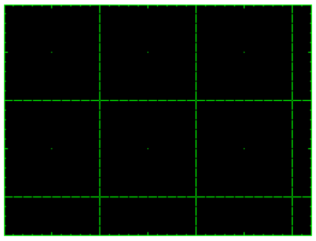
SHOW [MAIN|PIP1|PIP2|MEM:addr] [FILL|GRID] [ANI] [H:n] [W:n] [LW:width] [G:pixels] [D:delay] [R] [SW:width] [SO:offset] [FG:color] [BG:color] [N:frames] [ST:time] [ALPHA:n] [ROT:n] [F:n]

Writes a test pattern image into display memory. There are three test pattern images as shown below.



Default
(horizontal bars)


FILL


GRID

The test patterns rely on current register settings and may not display correctly if the registers are not configured properly. This command is useful for displaying an image during testing or evaluation. By using the same test image, the effect of a register change can be observed.

Where:

MAIN|PIP1|PIP2|MEM:addr
The window or memory address to write the test pattern to. If a memory *addr* is specified, the image is written to the specified address using the current display stride. If no window or memory address is specified, the SHOW command writes the test image to the MAIN window.

FILL|GRID	Specifies the test pattern to display. Test patterns are as shown above. If neither FILL or GRID is specified the default pattern is used.
ANI	Animates the test pattern until a key is pressed. During animation, press the SPACE bar to enter frame-step mode. Press any other key to exit frame-step mode and resume animation.

W:pixels	Width of the test pattern image, in pixels, specified as a decimal value.
H:lines	Height of the test pattern image, in lines, specified as a decimal value.
LW:pixels	Width of lines and borders for the GRID pattern, specified as a decimal value. (default = 1)
G:pixels	Pattern movement per frame during animation, specified as a decimal value. (default = 4)
D:ms	Delay between memory writes, in microseconds, specified as a decimal value. (default = 0)
R	Updates the start address, offset, and LUT registers automatically when appropriate.
SW:width	Rectangular mode Source Width specified as a decimal.
SO:offset	Rectangular mode Source Offset specified as a decimal. To enable rectangular mode only SO needs to be specified.
FG:color	Foreground color of the GRID, in RGB, specified as a hexadecimal value. (default = 00FF00)
BG:color	Background color of the GRID, in RGB, specified as a hexadecimal value. (default = 000000)
N:frames	Number of frames to show during animation before stopping specified as a decimal value. (default = stop by user)
ST:ms	Time to show animation before stopping. (default = stop by user).
Alpha:n	The alpha value for ARGB formats, specified as a hexadecimal value.
ROT:n	Rotates the test pattern 0, 90, 180, or 270°. N must be one of 0, 90, 180, or 270.
F:n	Data input format. If this parameter is not specified, PLAY will use the current register settings of the specified window to determine the format.

[0] RGB 332	[9] ARGB 1555
[1] RGB 565	[10] ARGB 4444
[2] Reserved	[11] ARGB 8565
[3] RGB888 Unpacked	[12] ARGB 8888
[4] YUV 422	[13] YUV422 DigitalOut 32
	[14] YUV422 DigitalOut 16

W[8|16|32] [StartAddr [Data1 Data2 Data3 ...]
Writes a given data sequence to memory starting at the specified start address.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last W command. If no previous W command has been issued, the unit size defaults to 16-bit.
StartAddr	The start address in memory to write data to. When StartAddr is specified, the memory write will begin from this offset. If StartAddr is not specified, the memory dump will begin where the previous Write ended. If a period "." is specified for the start address, the write is done at the same start address as used in the last W command.
Data#	The data value(s) to write to memory. Data values can be specified as text, numbers, or combinations of both. Numbers are assumed to be hexadecimal values unless otherwise specified with the correct suffix (binary = i, octal = o, decimal = t, hexadecimal = h). For example, 101i = 101 binary. To specify text values, enclose text data inside double quotes. If no data values are specified, the Write command enters ***modify*** mode. In modify mode the address and current data are displayed. The user is prompted to enter new data values. While in this mode, the user can type any of the following: - A new data value (in hex) followed by <ENTER> or <SPACE>. The write address will advance to the next address. -Press <ENTER> or <SPACE> to move to the next memory location changing the current address. - <MINUS> to move to the previous memory location - <Q> (or non-edit key) to leave MODIFY mode.

X[A|B] [8|16|32] [Index [Data]]
Reads/Writes the register at the specified index. If no data is specified, the specified register is read and the contents are displayed.

Where:

A	Displays all registers and their contents.
B	Displays all registers and their contents in a brief format.

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last X command. If no previous X command has been issued, the unit size defaults to 16-bit.
Index	The register index (in hex) to read from or write to. If no index is specified, the X command reads from the last specified X command index.
Data	Data value to write to the specified register index. Data values can be specified as text or numbers. Numbers are assumed to be hexadecimal values unless otherwise specified with a suffix (binary = i, octal = o, decimal = t, hexadecimal = h). For example, 101i = 101 binary. To specify a text value, enclose text data inside double quotes.

XF[B][8|16|32] Index Cycles Data1 [Data2 Data3 ...]
Fills the specified register index with the given data for the specified number of cycles.

Where:

B	Use burst mode. When burst mode is specified, the register fill sequence looks like (index, data, data, data ...). When burst mode is not specified, the register fill sequence looks like (index, data, index, data, index data …).
8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last XF command. If no previous XF command has been issued, the unit size defaults to 16-bit.
index	The register index (in hex) to fill.
cycles	The number of cycles to repeat the data pattern (in hex). To cycle forever, set cycles to 0.
data#	The data value(s) to fill the register with. If multiple values are given, then the pattern will repeat for the specified number of cycles. Data values can be specified as text, numbers, or combinations of both. Numbers are assumed to be hexadecimal values unless otherwise specified with the correct suffix (binary = i, octal = o, decimal = t, hexadecimal = h). For example, 101i = 101 binary. To specify text values, enclose text data inside double quotes.

XR[B][8|16|32] Index [Cycles]
Reads data from the specified register index for a specified number of cycles.

Where:

B	Specifies burst mode. When burst mode is specified the register read sequence looks like (index, read, read, read …). When burst mode is not specified the read sequence looks like (index, read, index, read, index, read …).
8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last XR command. If no previous XR command has been issued, the unit size defaults to 16-bit.
index	The register index to read data from (in hex).
cycles	The number of cycles to read data from the specified register index (in hex). To cycle forever, set cycles to 0.

?|HELP [RegIndex|SearchText]
Displays the Help screens. The help screens contain a summary of all available commands and a script language reference. If a parameter is given, Help can provide context sensitive information on a given register or keyword. This is useful for searching for specific chip information, or which registers/features are referenced by a specific command.

Where:

RegIndex	A 13513 register index (in hex) for which more information is needed.
SearchText	A keyword that will be searched for in the help data. (e.g. VNDP or BitBLT)

3.2. Advanced Play Commands

PLAY also includes advanced commands which are designed to perform a specific function. The commands included in this group are summarized in the following table.

Command	Description
CALC Expression	Provides built-in calculator functions
DEBUG	HAL debugging
DIAGNOSE	Diagnose using specification restrictions
DRAW [MAIN1|PIP1|PIP2] [TEXT|LINE|BOX|RECT] [...]	Draws various predefined graphics to the screen
FDUMP[8|16|32] Filename Area Startaddr [Endaddr|Len] [options..]	Writes data to a file
FLOAD[8|16|32] Filename Area Startaddr [Endaddr|Len] [options..]	Reads data from a file
FLOG Filename [CMD|OUT|SET:ON|OFF] [...]	Logs output to a file
FSIZE[8|16|32] Filename	Returns the file size of the specified file
H [Lines]	Halts after the specified number of lines
L[1|2][A] Index [[Red Green Blue]|[Data]]	Reads or Writes the Look-up Tables (LUT)
LFT [ON|OFF]	Controls LCD frame transfer mode
PLL[1|2] [freq]	Configures the PLL for specified frequency
PS [ON|OFF]	Controls power save mode
[RUN] [Scriptfile]	Runs the specified script file
RUNPATH [Path1[Path2 Path3 .. PathN]]	Sets the path where script files can be run from
SS	Synchronizes the strides
VER	Displays the build version information

CALC Expression
Provides a built-in calculator function that can calculate the specified expression.

Where:

expression	The expression to process. (see Section 5 for a discussion of PLAY expressions)

DEBUG
Toggles the state of the HAL debugging flag. When this flag is enabled, the HAL displays all register writes except for LUT and I2C programming. Register reads are not displayed.

DIAGNOSE
This command is used to diagnose deviations from the recommended S1D13513 specifications for the register and feature settings. Diagnose outputs a report of all deviations.

For example if a register is set to a value that is out the allowable range according to the S1D13513 Hardware Specification, a message is included in the Diagnose report.

DRAW [MAIN|PIP1|PIP2] [TEXT|LINE|BOX|RECT ..]
Draws various graphics into display memory. Note that range checks, boundary checks, and clipping are not performed by the DRAW command. The parameters required by the DRAW command varies based on the type of graphic to be displayed. There are four types of graphics supported as shown below.

```
DRAW [MAIN|PIP1|PIP2] TEXT topLeftX topLeftY fg bg string
DRAW [MAIN|PIP1|PIP2] LINE topLeftX topLeftY btmRightX btmRightY color
DRAW [MAIN|PIP1|PIP2] BOX topLeftX topLeftY width height color
DRAW [MAIN|PIP1|PIP2] RECT topLeftX topLeftY width height color
```

Where:

MAIN|PIP1|PIP2	The window where the graphic is drawn. If this parameter is not specified, the graphic is drawn in the window specified in the last DRAW command.
TEXT	Draws text on the display using an 8pt monospace font.
LINE	Draws a simple line on the display.
BOX	Draws an outlined rectangle on the display.
RECT	Draws a solid rectangle on the display.
topLeftX	The horizontal position of the top left corner of the graphic, in pixels, relative to the MAIN/PIP window.
topLeftY	The vertical position of the top left corner of the graphic, in pixels, relative to the MAIN/PIP window.
btmRightX	The horizontal position of the bottom right corner of the graphic, in pixels, relative to the MAIN/PIP window.
btmRightY	The vertical position of the bottom right corner of the graphic, in pixels, relative to the MAIN/PIP window.
fg, bg	The foreground (fg) and background (bg) colors for the text string. If the foreground and background colors are the same, the text is transparent.
string	The text to be displayed. If the foreground and background colors are the same, the text is transparent.

color	The 32-bit ARGB or 24-bit YUV color, in hex format. The value must be prefixed with either R or Y to indicate RGB or YUV. If the value is not prefixed, the value is assumed to be RGB.
width	The width of the BOX or RECT to be displayed, in pixels.
height	The height of the BOX or RECT to be displayed, in pixels.

FDUMP[8|16|32] Filename Area StartAddr [EndAddr|Len] [options..]
Write the contents of register, memory or system memory to a file.

Where:

8|16|32	The unit size (8/16/32-bit) for the command. If the unit size is not specified, the command uses the unit size specified in the last FDUMP command. If no previous FDUMP command has been issued, the unit size defaults to 8-bit.
Filename	The name of the file where the binary register/memory dump is stored. To read the data but not write to the file, specify the filename as NUL. If Filename already exists, the output can be appended to the existing file or can overwrite the existing file based on the setting of the A option.
Area	The location where memory is read from. MEM – Binary read of the display memory. REG – Binary read of the registers. SYS – System-wide binary read.
StartAddr	The starting address to read data from. If a period "." is specified for the start address, the dump is done from the same start address as used in the last FDUMP command. StartAddr must be specified.
Endaddr|Len	The end address to read data from. The end address can be specified as a physical memory address or as the length (in terms of unit size) of the binary dump. If a length is specified, it must be prefixed with "L". For example, specifying L8 when the unit size is 16-bit causes the FDUMP command to dump 8 words from the starting address. If an "L" is not specified, PLAY assumes that the value specified is a physical memory address.

Options	Additional options: A:ON|OFF - Append to file (default is OFF). B:ON|OFF - Burst mode (default is ON). N:n - Number of times to repeat writing data (dec). D:n - Delay in microseconds between reads (dec).

Examples:

```
FDUMP8 memory.bin MEM 0 L10000 A=ON          'Dump display memory
FDUMP fifodump.bin REG 123 L1 N=16t D=10     'Dump FIFO from REG[123]
FDUMP16 system.bin SYS 50000 60000           'Dump system memory
```

Note
Arguments after FDUMP must be Filename, Area, StartAddr, EndAddr|Len, in that order. Optional arguments can follow any sequence.

FLOAD[8|16|32] Filename Area StartAddr [EndAddr|Len] [options..]
Reads register or memory contents from a specified file.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last FLOAD command. If no previous FLOAD command has been issued, the unit size defaults to 16 bit.
Filename	The name of the file to read the register or memory contents from.
Area	The location where the register or memory contents are written to. MEM – Binary write of the display memory. REG – Binary write of the registers. SYS – System-wide binary write. ELF – ELF file write to display memory.
StartAddr	The starting address in memory to write data to. If a period "." is specified for the start address, the data is written to the same start address as used in the last FLOAD command.

EndAddr|Len	The end address in memory to stop writing data to. The end address can be specified as a physical memory address or as the length (in terms of unit size) of the memory write. If a length is specified, it must be prefixed with “L”. For example, specifying L8 when the unit size is 16-bit causes the FLOAD command to write to memory starting from the start address with 8 words. If an “L” is not specified, PLAY assumes that the value specified is a physical memory address.
options	Additional options: A:ON|OFF - Load all frames consecutive (default=OFF). Alpha:n – Alpha value for BMP/PPM image (hex). B:ON|OFF - Burst mode (default is ON). D:Delay - Delay in microseconds between writes (dec). EOF - Load up to end-of-file. Used with REG option. It will override any value specified for N=. F:format – Data format. (required if area = MEM|REG & file = BMP|PPM. [0] RGB 332 [9] ARGB 1555 [1] RGB 565 [10] ARGB 4444 [2] Reserved [11] ARGB 8565 [3] RGB 888 [12] ARGB 8888 [4] YUV 422 [13] YUV 422 DigitalOut FPS - Show frame-per-second calculation on exit. FR:fps - Set frame rate of streaming video in frames-per-sec. FRM:n - Number of frames to load successively (dec). L:loops - Number of loops. Used with A=ON. Use L:0 to loop until user presses key. N:times - Number times to repeat writing data (dec). O:offset - Byte offset in file to begin loading (hex).

Note

The arguments after FLOAD must be Filename, Area, StartAddr, EndAddr/Len (in that order). Optional arguments can follow in any sequence.

Examples:

```
FLOAD8 memory.bin MEM 1000 1FFF F=1          'Load to display memory
FLOAD fifoload.bin REG 123 L1 N=16t D=10     'Load FIFO to REG[123]
FLOAD32 system.bin SYS 0 10                  'Load to system memory
```

FLOG Filename [CMD:ON|OFF] [OUT:ON|OFF] [SET:ON|OFF]
Logs PLAY activity to a file. This includes commands and displayed data. This feature is useful for maintaining a record of steps which can be reviewed.

Where:

Filename	The name of the file that output is logged to. If the file already exists then output is appended to the file. To stop logging, set the Filename to NUL.
CMD:ON|OFF	Determines whether command prompt output is logged.
OUT:ON|OFF	Determines whether general output and output from PRINT commands are logged.
SET:ON|OFF	Determines whether output generated from SET commands are logged.

Examples:
```
FLOG log.txt CMD:OFF OUT:ON
FLOG log.txt SET:OFF
```

FSIZE[8|16|32] Filename
Returns the size of the specified file in terms of the unit size. This is most useful for scripts that need to assess the size of a file before continuing. The file size returned by FSIZE can be assigned to a variable as in the examples below.

Where:

8|16|32	The unit size (8-bit|16-bit|32-bit) for the command. If a unit size is not specified, the command uses the unit size specified in the last FSIZE command. If no previous FSIZE command has been issued, the unit size defaults to 16-bit.
Filename	Name of the file to be examined.

Examples:
```
$var = fsize file.txt
$var = fsize file.bin
```

H [Lines]
This command sets the number of lines of text that are displayed before halting (pausing) for user input. The Halt count is reset every time a command prompt is displayed so only long data listings are affected. Halt is useful when large blocks of memory are being dumped and the user wishes to stop after each screen is filled.

Where:

Lines	The number of lines that are shown before halting the displayed data (in decimal). If this value is set to 0, the display will never halt.

L[1|2][A] Index [[Red Green Blue]|[Data]]
Reads/writes the red, green, and blue Look-up Table (LUT) components. If the red, green, and blue components are not specified, the Look-up Table for the given index is read and the RGB elements are displayed.

Where:

1	Selects LUT1 - Gamma 8-bit RGB Look-up Table.
2	Selects LUT2 - BitBLT 16-bit Look-up Table.
A	When specified this option causes the function to display all the current LUT values. Index, red, green, blue, and data should not be included on the command line.
Index	Index into the LUT (in hex).
Red	Red component of LUT1[index] (in hex).
Green	Green component of LUT1[index] (in hex).
Blue	Blue component of LUT1[index] (in hex).
Data	16-bit data value of LUT2[index] (in hex)

Examples:
```
L1A                 ' Dump contents of the Gamma LUT
L2A                 ' Dump contents of the BitBLT LUT
L1 0C FF FF FF      ' Write Gamma LUT RGB components at index 0Ch
L2 10 C35A          ' Write BitBLT LUT at index 10h
L2 10               ' Read BitBLT LUT at index 10h
```

LFT [ON|OFF]
Sets the automatic LCD Frame Transfer mode. Some panel technologies are not actively updated. These panel types require the frame data be transferred to them. To make testing easier this feature enables PLAY to transfer data to the panel after each display memory update. If neither ON or OFF is specified then the current LFT mode is displayed.

Where:

ON	Enables LCD Auto Frame Transfers after every command is executed.
OFF	Disables LCD Auto Frame Transfers.

PLL[1|2] [Freq]
Sets the indicated PLL to the specified frequency. If no parameter is specified, the current PLL frequency is displayed.

Where:

1	Selects PLL1
2	Selects PLL2
Freq	Frequency (in MHz) to program the PLL to. A frequency in KHz can be specified by appending the suffix “KHz”. If spaces are used, the string must be placed within double quotes.

Examples:

pll 40.034	Sets PLL to 40.034 MHz
pll 40KHz	Sets PLL to 40 KHz
pll "40 KHz"	Sets PLL to 40 KHz
pll $var	Where $var=50t
pll $var	Where $var="32.567 MHz"

PS [ON|OFF]
Controls the Power Save Mode and starts the power on/off sequence.

Where:

ON	Enables power save mode and starts the power down sequence.
OFF	Disables power save mode and starts the power up sequence.

[RUN] [Scriptfile]
Opens the specified script file and executes each line as if it was typed from the command prompt.

Typing "RUN" and suffixing the Scriptfile with ".txt" is optional. When PLAY does not recognize a command, it will assume the entry refers to a script file and will attempt to run that script. If PLAY cannot locate the scriptfile, then ".txt" will be appended and a second attempt to open the file will be made.

Note
If a script file named "play.txt" is created, it will be loaded when PLAY first runs.

Where:

Scriptfile	File name of the text scriptfile.

RUNPATH [Path1 [Path2 Path3 …]]
Sets the directory path(s) from where PLAY will search for script files. Up to 8 paths may be specified.

Where:

Path#	The path(s) where script files are located. Directory paths that contain spaces must be enclosed in double quotes.

Example:
RUNPATH c:\ ..\scripts "D:\work\My Play Scripts"

SS
Synchronizes the strides of all windows based on the current width, height, and bpp.

VER
Shows build version information and PLAY configuration information.

4. Preprocessing

PLAY provides the ability to preprocess input from the command prompt or script file before interpreting the various commands, statements, and expressions available in the language.

The interpreter can perform radix and type conversions using an internal converter. To specify the desired conversion, append the following construct to the expression.

```
.type[radix][width]
```

Where:

type	Specifies the format that the token will be converted to (“s” = string, “n” = number).
radix	Specifies the numeric base that the token will be converted to (“h” = hexadecimal, “t” = decimal, “o” = octal, “i” = binary).
width	The length, in number of characters, to which the resulting string will be extended if it is not already at least that length. If the length is prefixed with a 0, the string is lengthened with zeroes instead of spaces.

Note

All numerical input at the command line or prompt is assumed to be hexadecimal unless otherwise stated in the command description. If a number must be forced to a specific numerical base, append one of “h” (hexadecimal), “t” (decimal), “o” (octal), or “i” (binary) directly to the end of the number.

5. Script Files

PLAY can be controlled with script files. This is useful when there is no display to monitor command keystroke accuracy, or when various registers must be quickly changed to view results. It can also ease the tedious nature of repetitive testing by automating much of the work.

A script file is an ASCII text file. Script files can be executed from within PLAY using the RUN command (i.e. = run dumpregs.scr). Alternately, the script file can be executed from the OS command prompt. On a PC platform, a typical script command line might be:

```
PLAY run dumpregs.scr > results
```

This causes the script file dumpregs.scr to be interpreted as commands by PLAY with the results redirected to the file results.

A powerful feature of script files is their ability to act like functions. This is accomplished by allowing parameters to be passed to scripts upon execution, and by allowing scripts to return values upon completion.

5.1. Expression and Token Usage within Scripts

The use of expressions and tokens in scripts provides a powerful method for performing automated calculations. Expressions are constructs consisting of tokens and atomic operators that are computed before the execution of a command. Tokens make up the basic atomic unit for the preprocessor and can consist of simple numeric input, strings, or user defined variables.

5.1.1. Comments

All comments must be preceded by a single quote (’) and may begin anywhere on a line. All characters after the quote on the same line are not executed by PLAY.

5.1.2. Expressions

An expression may consist of a single token or a combination of expressions using the following constructs. Expr1, expr2, and expr3 denote separate numeric expressions.

Arithmetic:

expr1**expr2	Returns the exponential expansion of expr1 by multiplying expr1 with itself expr2 times.
expr1*expr2	Returns the numeric product of expr1 and expr2.
expr1/expr2	Returns the numeric dividend of expr1 by expr2.
expr1%expr2	Returns the numeric modulus of expr1 by expr2 (i.e. the remainder when expr1 is divided by expr2).
expr1+expr2	Returns the numeric sum of expr1 and expr2.
expr1-expr2	Returns the numeric difference of expr1 and expr2.

Bit Operators:

~expr1	Returns the bitwise complement of expr1 (i.e. returns the result of inverting the bits of expr1).
expr1&expr2	Returns the bitwise And of expr1 and expr2 (i.e. returns 1 if there exists a 1 in the bit representations of both expr1 and expr2).
expr1|expr2	Returns the bitwise Or of expr1 and expr2 (i.e. returns 1 if there exists a 1 in the bit representations of either expr1 or expr2).
expr1^expr2	Returns the bitwise Exclusive-Or of expr1 and expr2 (i.e. returns 1 if there exists a 1 in the bit representations of either expr1 or expr2, but not in both).
expr1<<expr2	Returns the bits of expr1 shifted left by expr2 counts (i.e. returns the result of moving all of the bits in expr1 left expr2 times, dropping the expr2 left-most bits).
expr1>>expr2	Returns the bits of expr1 shifted right by expr2 counts (i.e. returns the result of moving all of the bits in expr1 right expr2 times, dropping the expr2 right-most bits).

Logical:

!expr1	Returns the logical NOT of expr1 (i.e. if expr1 equates to true it returns false and if expr1 equates to false it returns true).

expr1&&expr2	Returns the logical AND of expr1 and expr2 (i.e. if expr1 equates to true and expr2 equates to true then it returns true, otherwise it returns false).
expr1||expr2	Returns the logical OR of expr1 and expr2 (i.e. if either expr1 or expr2 equates to true then it returns true, otherwise it returns false).
expr1ANDexpr2	Returns the logical AND of expr1 and expr2 (i.e. if expr1 equates to true and expr2 equates to true then it returns true, otherwise it returns false).
expr1ORexpr2	Returns the logical OR of expr1 and expr2 (i.e. if either expr1 or expr2 equates to true then it returns true, otherwise it returns false).
expr1XORexpr2	Returns the logical XOR of expr1 and expr2 (i.e. if either expr1 or expr2 equates to true, but not both, then it returns true, otherwise it returns false).

Relational:

expr1==expr2	Returns true if expr1 is numerically or logically (true and false) equal to expr2 and returns false otherwise.
expr1!=expr2	Returns true if expr1 is numerically or logically (true and false) not equal to expr2 and returns false otherwise.
expr1<expr2	Returns true if expr1 is numerically less than expr2 and returns false otherwise.
expr1>expr2	Returns true if expr1 is numerically greater than expr2 and returns false otherwise.
expr1<=expr2	Returns true if expr1 is numerically less than or equal to expr2 and returns false otherwise.
expr1>=expr2	Returns true if expr1 is numerically greater than or equals to expr2 and returns false otherwise.

Other:

expr1?expr2:expr3	Returns the result of a conditional If-Then-Else. If expr1 is true it will return expr2, otherwise it will return expr3.

Notes:

1. As in many programming languages, TRUE corresponds with any value greater than zero. FALSE corresponds with any value less than or equal to zero. PLAY will return 1 for TRUE values and 0 for FALSE values.
2. Precedence in execution does not always work as expected. To force a certain order of precedence enclose all clauses in parenthesis. Parenthesis may be nested.

5.1.3. Statements

Statements consist of all viable PLAY related commands, including the aforementioned command listings and the following scripting specific commands. Statements can consist of several sub-statements delimited by semicolons. An end of line will also denote the end of the statement (i.e. statement; statement; statement).

5.1.4. Tokens

Tokens denote the various forms of data that PLAY can manipulate, and can be delimited in the following forms:

Arguments and Other Values

arg[n]	When a script is provided with arguments at execution, the arguments are stored in this array. Each argument is defined by a sequentially increasing number (represented by ‘n’). Argument ‘0’ is the name of the script.
argn	Contains the total number of arguments in the script.
retval	The value assigned to this token is passed to whatever function or expression that called the script, allowing for return values.
rand	A random number generator that returns a 32-bit unsigned integer.

Display Buffer, Memory, and Register Access

reg[val]	References the register at location ‘val’, where val is a token representing the memory location, or a token representing the memory location with a relative offset.
data8[loc]	References the 512K hardware data repository at the 8-bit scalar index ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.
data16[loc]	References the 512K hardware data repository at the 16-bit scalar index ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.
data32[loc]	References the 512K hardware data repository at the 32-bit scalar index ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.

disp8[loc]	References the display buffer at the 8-bit offset ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.
disp16[loc]	References the display buffer at the 16-bit offset ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.
disp32[loc]	References the display buffer at the 32-bit offset ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.
sys8[loc]	References the system-wide memory at the 8-bit offset ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.
sys16[loc]	References the system-wide memory at the 16-bit offset ‘loc’, where loc is a token representing the| memory location, or a token representing the memory location with a relative offset.
sys32[loc]	References the system-wide memory at the 32-bit offset ‘loc’, where loc is a token representing the memory location, or a token representing the memory location with a relative offset.

Features and Bitfields

bpp, hdp, lcd2hdp, paneltype, swivelview	Denotes features and bitfields built into PLAY.

Variables

$variable	Denotes a variable, where ‘variable’ can be a text string or unsigned 32-bit integer. All variables must be prefixed by the ‘$’ character when used.

Note

PLAY has no concept of scoping. Using the same variable name in a script called from a parent script will overwrite the value in the parent variable.

5.2. Script Commands

Script commands are PLAY commands that are specifically for use within script files.

Command	Description
DELAY time	**Delays script execution for a specified number of microseconds**
ELSE statements	**Contrary condition for the conditional IF construct**
IF expression [THEN] statements	**Conditional IF construct**
INPUT LINE|KEY|PEEK	**Returns input from the keyboard**
GOTO sectionname|expression	**Jumps to a marked section of code**
PRINT "**m**essage"|token|**expression [...]**	**Prints text messages (including variables, tokens, or expressions that can be resolved to text)**
SECTION sectionname	**Marks a section of code within a script**
SHELL ["command" ["args"] [...]]	**Runs a system command shell**
SINGLESTEP [ON|OFF]	**Steps through the command in a script file line by line**
SLEEP time	**Causes the script to go to sleep for a specified number of milliseconds**
VERBOSE [CMD|OUT|SET:ON|OFF] [...]	**Controls whether output is displayed for various commands**
WHILE expression [DO] statements	**Conditional WHILE construct**

Delay time
Delays script execution for the specified length of time, in microseconds. This command is useful for short delays as it uses the internal timer and doesn't release the CPU during the delay. The Sleep command should be used for longer delays.

Where:

time	The number of microseconds execution of the script will be delayed. This value is a decimal number.

Example:
```
delay 5                    'Delay for five microseconds
```

ELSE statement
Contrary condition for use with the conditional IF construct. This construct is used to direct program flow within the script when the expression evaluated in the IF construct is FALSE.

Note
ELSE and the following execution statement must be contained within one line.

Where:

statement	The statement to execute if the IF expression is FALSE.

IF expression [THEN] statement
Conditional IF construct used to direct program flow based on the specified expression. If the expression is TRUE, then the statement is executed. The ELSE command can be used in conjunction with the IF command.

Note
The IF statement and the following execution statement must be contained within one line.

Where:

expression	The expression which is evaluated in order to direct program flow. If the value of the expression is greater than zero, or true, the script will execute the specified statement.
statement	The statement to execute if the IF expression is TRUE.

Example:
The following script will break out of a repeating delay loop after 5 repetitions.

```
‘Set the variable “key” to 0
set $key 0

Section LOOP
        delay 5
        ‘Add one to key
        set $key $key+1
        ‘Repeat if the variable key is less than five
        if $key<5 Then Goto LOOP
        ‘Otherwise go to the END section
        Goto END
Section END
```

INPUT LINE|KEY|PEEK
Returns user input from the keyboard into the provided variable ($var). KEY waits for a single keystroke. LINE waits until the user presses the <ENTER> key. PEEK is similar to KEY, but returns immediately (with $var) if the user has not pressed a key. If the user did press a key, the value is contained in $var.

Where:

LINE	Records all user input until the end of line character is received.
KEY	Records a single key from the user.
PEEK	Records the last key pressed by the user if the user has pressed a key. If the user has not pressed a key, then the provided variable will be empty.

Example:
The following script will break out of a repeating delay loop if the user enters a key.

```
set $key 0

Section LOOP
        delay 5
        ‘Look for input without pausing
        $key = input PEEK
        if $key Then Goto END
        Goto LOOP
Section END
```

GOTO sectionname|expression
Goes to the section of the script marked by the specified sectionname.

Where:

sectionname	The section where the script will continue execution. The section name may be held in a variable.

Examples:
The following script will delay program execution for 5 microseconds, then delay again repeatedly.

```
Section LOOP
        delay 5                 ‘Delay for five microseconds
        goto LOOP               ‘Go back to the start of the section
Section END
```

PRINT string|token|expression [...]
Prints a string, token, or any expression that evaluates to text. C style delimiters can be used to format the text. For example, to wrap to the next line use “\n”. Any valid expression within {braces} will be processed and substituted by the results.

Where:

string	The string to be printed.
token	The string token that will be displayed.
expression	The expression, which must evaluate to text, that will be printed.

Examples:
The following script will print the number of microseconds that passes until the user enters a key.

```
set $key 0
set $microSeconds 0
Section LOOP
    delay 5
    set $microSeconds $microSeconds+5
    ‘print the time passed at the start of the line and after converting it to a string
    ‘in base ten notation
    print “\r” $microSeconds.st
    $key = input PEEK
    if $key Then goto END
    Goto LOOP
Section END
```

The following lists some other examples of using the print command.

```
print "Hello world\n"                  "Hello world" (linewrap)
print $usertext                        "hi mom"
print $var.s                           "4"
print $var.s10                         "0000000004"
print 3c.s                             "3C"
print 3c.st                            "60"
print 3c.sh4                           "  3C"
print 3c.sh04                          "003C"
print REG[0104h].sh4 "h"               "1234h"
print "coords=" $x.st "," $y.st "\n"   "coords=4,6" (linewrap)
print "coords={$x.st},{$y.st}\n"       "coords=4,6" (linewrap)
```

SECTION sectionname
Marks the start of a section of code with the given sectionname. The GOTO command may be used to direct program flow to a specific section. Sections work in a similar manner to labels in the C and assembler syntaxes, where a section is a location to which execution may be directed.

Where:

sectionname	The name of the new section.

SHELL [“command” [“argument”] [...]]
Executes a system command shell and returns to play once the shell exits, or when the (optional) provided command completes.

Where:

command	The system command to be executed.
argument	The argument to be passed to the system command.

Example:

```
shell “cls”                  ‘ CLS clears the PC display
```

SINGLESTEP [ON|OFF]
Determines how script files are run. When SINGLESTEP is ON, script files are executed one line at a time. In this mode, the user must proceed through a script manually by pressing a key to execute each command.

Where:

ON|OFF	Toggles single line execution of script files on or off.

SLEEP time
Sleeps for the specified length of time, in milliseconds. This command releases the CPU during the sleep time and should be used for longer delays. For short delays, the Delay command can be used.

Where:

time	The number of milliseconds to sleep, in milliseconds (decimal)

VERBOSE [CMD:ON|OFF] [OUT:ON|OFF] [SET:ON|OFF]
Controls the level of output sent to the display. If no parameters are entered, the VERBOSE command displays the current settings.

Where:

CMD:ON|OFF	Controls verbose output for command line and prompt messages.
OUT:ON|OFF	Controls verbose output for the output generated by most commands.
SET:ON|OFF	Controls verbose output for the output generated by the SET command.

Example:
The following script clears the screen without notifying the user.

```
‘The following line turns off all output from the interpreter
verbose cmd:off out:off set:off
shell “cls”                     ‘ CLS clears the PC display
```

WHILE expression [DO] statements
Boolean function used to direct program flow based on the evaluation of a specified expression.

Note
The WHILE command, expression, and statement must be contained within one line.

Where:

expression	The expression which is evaluated to direct program flow. While the value of the expression is TRUE (greater than zero), the program will repeatedly execute the provided statement.
statement	The statement to execute while the expression is TRUE.

Example:
The following script will delay execution until the user presses a specific key.

```
‘Set key to be an empty string. Numerical and string variables cannot be compared
set $key “ “
‘Repeat while the key pressed is not the Escape key
‘Note how the statement the while loop executes is a compound statement. Such
‘statements must be contained on the same line as the while loop
while $key != “ESC” Do delay 5; $key = input PEEK
```

6. Comments

- All displayed numeric values are considered to be hexadecimal unless identified otherwise. For example:

  - 10 = 10h = 16 decimal.

  - 10t = 10 decimal.

  - 010‘b = 2 decimal.

- Redirecting commands from a script file allows those commands to be executed as if entered by a user.

7. Change Record

X78B-B-001-01 Revision 1.02 - Issued: March 29, 2018

- Updated address/contact page
- Updated Epson web page and email address
- Minor formatting changes

X78B-B-001-01 Revision 1.01 - Issued: December 21, 2007

- minor edits and technical updates

X78B-B-001-01 Revision 1.0 - Issued: September 21, 2007

- section 2, added the /nocard switch and noted that the arrow keys can be used to scroll through the command history
- section 4.1, modified the Show command syntax
- section 5.2, improved the Delay script command description
- section 5.2, improved the Sleep script command description
- additional minor edits

X78B-B-001-00 Revision 0.01 - Issued: June 18, 2007

- initial draft of the document

8. Sales and Technical Support

For more information on Epson Display Controllers, visit the Epson Global website.

https://global.epson.com/products_and_drivers/semicon/products/display_controllers/

For Sales and Technical Support, contact the Epson representative for your region.

https://global.epson.com/products_and_drivers/semicon/information/support.html