HSP-8 PROGRAMMER'S MANUAL

HSP-8 Communications device driver
32 bit Dynamic Link Library
For use with Windows 95, 98, NT v4.0, 2000
VERSION 2.0

The HSP-8, an Overview

Probe Product Corporation's HSP-8 is a high speed signal conditioning data processing system designed to allow direct connection of gaging and displacement transducers to an IBM compatible computer.

The HSP-8 lends itself to dynamic gaging, since the on-board co-processor can complete complex functions, with results being passed to the host computer.

Installation
Refer to the HSP-8 installation manual to install the drivers for the operating system you will be using.

Files Required to Write 32 bit C Programs that use HSP-8:

Hspcomm.h   The 32bit API function declarations.
Hspcomm.lib   Link this to your application.

HSPCOMM.DLL Software Interface

Overview

The HSP Communications device driver (hspcomm.dll) is written in "C" and compiled as a 32 bit Dynamic Link Library (dll) file for use with Windows 98/NT/2000. Although the function names are the same, the device drivers for Windows NT and Windows 98 are different so the target platform should be known or defined before using the driver.

To use the HSP Communication functions in your software, include the supplied hspcomm.h file in the C source files. Link the file hspcomm.lib with your application. The application must be able to locate the hspcomm.dll file at run time.

Typical use:

```
//STEP 1 - Initialize
  int HspLoadStatus;

  //open hsp driver
  HspLoadStatus = HspCommInit(0);
  if(HspLoadStatus==0){
    HspSetScanTime(100);                // scan time set to 10.0 ms
          // scan time must be set or it will default
          // to zero (meaning no scanning)
    HspStartScan();
  }

//STEP 2 - define channels
HspDefineChannel(1,"t1+t2");
HspDefineChannel(2,"max(t1+t2)");
HspDefineChannel(3,"min(t1+t2)");

//STEP 3 - read values
float values[3];

HspReadChannel(1,&values[0]);          // pass adr of space for float
HspReadChannel(2,&values[1]);          // pass adr of space for float
HspReadChannel(3,&values[2]);          // pass adr of space for float

// display or store values

// to reset max/min/tir functions used in formulae call HspResetMM()
if(ResetPushedMessage)
  HspResetMM();
```

Communication Functions

Overview

There is one function for each of the HSP-8 commands. To use the functions, call the function with its arguments. The function will issue the proper command code and arguments to the HSP-8. The function will return the HSP-8's response.

Status Code

All HSP-8 functions return a status code that indicates the success or failure of the command execution. The status code is returned as the value of the function. The functions that return values in addition to the status code require arguments that are a pointer to a place to put the returned value. The following rules pertain to all functions:

All functions return an integer which will be:

zero	If the command was completed successfully.
-1	If unable to communicate.
+value	If the command was not executable. (see Error Codes in Appendix C)

General Arguments

When functions require a channel number as an argument the channel number must be between 1 and 32. When functions require a transducer number as an argument the transducer number must be between 1 and 32. (Transducer numbers greater than 8 will be accepted but are only meaningful if slave HSP-8s are installed.)

General Return Values

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Floating Point Arguments

This DLL only supports FP_IMODE_NATIVE, which is the default floating-point input mode. Changing the floating-point input mode with the HspSetFpIn function will cause unpredictable results.

Floating Point Return Values

This DLL only supports FP_OMODE_NATIVE, which is the default floating-point input mode. Changing the floating-point input mode with the HspSetFpOut function will cause unpredictable results.

Function HspCommInit

Description   Searches all attached LPT ports for an HSP-8. If found, opens a link to the HSP-8.

In C   int HspCommInit(int *dummy*);

In BASIC   Declare Function HspCommInit Lib "hspcomm" (_
                     ByVal *dummy* As Integer _
) As Integer

                    dummy   Not used.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.

Remarks   This function or HspCommInitEx must be the first function called before any of the other functions in HSPCOMM.DLL are used.

Function HspCommInitEx

Description   Searches only the specified LPT port for an HSP-8. If found, opens a link to the HSP-8.

In C   int HspCommInitEx(int *lpt*, int *dummy*);

In BASIC   Declare Function HspCommInitEx Lib "hspcomm" (_
  ByVal *lpt* As Integer _
  ByVal *dummy* As Integer _
) As Integer

Lpt   Lpt port where HSP-8 is attached.
dummy Not used.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.

Function HspMsgBox

Description	Controls the display of an error dialog box when an HSP-8 function returns an error.
In C	void HspMsgBox(int *enable*);
In BASIC	Declare Function HspMsgBox Lib "hspcomm" (_ ByVal *enable* As Integer _) As Integer *enable* Non-zero to enable the error dialog box.
Return Value	If successful, the function returns 0. Otherwise, it returns a nonzero value.
Remarks	The HspMsgBox function enables/disables a dialog box which appears whenever there is an error detected in an hspcomm command. The dialog box translates the interface error code into a brief descriptive error message. The enabling/disabling of the error dialog box does not affect the error codes returned by the other functions.

Function EppInstallMsg

Description   Controls the display of parallel port driver startup diagnostic messages. This is not generally useful, except as an aid in diagnosing problems with the parallel port communications.

In C   void EppInstallMsg(int *enable*);

In BASIC   Declare Function EppInstallMsg Lib "hspcomm" (_
  ByVal *enable* As Integer _
) As Integer

enable Non-zero to enable the error dialog box.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.

Remarks   The EppInstallMsg function enables/disables a series of dialog boxes that report on the progress of detecting the HSP-8 connected through a LPT port. The information displayed by these messages is useful in diagnosing the communication problems.

Function EppDiagMsg

Description   Controls the display of an error dialog box when the parallel port
                      communication driver returns an error.

In C   void EppDiagMsg(int *enable*);

In BASIC   Declare Function EppDiagMsg Lib "hspcomm" (_
                        ByVal *enable* As Integer _
) As Integer

                      enable Non-zero to enable the error dialog box.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.

Remarks   The EppDiagMsg function enables/disables a dialog box which appears
                      whenever there is an error detected in the communication with the HSP-8.
                      This dialog box provides more specific communication error descriptions than
                      the HspMsgBox, however, this dialog box is only for communication errors.
                      The dialog box translates the interface error code into a brief descriptive error
                      message.

                      The enabling/disabling of the error dialog box does not affect the error codes
                      returned by the other functions.

Function HspDLL

Description	Gets the version number of the Dynamic Link Library.
In C	int HspDLL(char *_VersionString_);
In BASIC	Declare Function HspDLL Lib "hspcomm" (_ ByVal _VersionString_ As String, _) As Integer
	VersionString Version number of the Dynamic Link Library.
Return Value	If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.
Remarks	_Preliminary_

Function HspStartScan

Description	Enables HSP-8 scanning of MAX/MIN/TIR elements used in channel formulae.
In C	int HspStartScan(void);
In BASIC	Declare Function HspStartScan Lib "hspcomm" (_) As Integer
Return Value	If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.
Remarks	None

Function HspStopScan

Description   Disables HSP-8 scanning of MAX/MIN/TIR elements used in channel formulae.

In C   int HspStopScan(void);

In BASIC   Declare Function HspStopScan Lib "hspcomm" (_
) As Integer

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   This function disables the scanning of formulae which contain MAX, MIN or TIR functions. The functions will be unaffected by changes in their arguments.

Function HspDefineChannel

Description	Defines an HSP-8 channel formula.
In C	int HspDefineChannel(int *channel*,char **formula*);

In BASIC	Declare Function HspDefineChannel Lib "hspcomm" (_
	  ByVal *channel* As Integer, _
	  ByVal *formula* As String _
	  ) As Integer

	channel A channel number. *formula* The formula string.
Return Value	If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.
Remarks	The argument *channel* must be between 1 and 96, inclusive. This function will define a formula for a channel. The *formula* argument is a pointer to a string that contains a valid formula. The formulae are supplied to the HSP-8 in the form of an ASCII string terminated by a null byte. See the description of formulae in the section **HSP-8 FORMULA SYNTAX** for details of allowed formats.

Function HspReadFormula

Description   Gets the formula string for the specified channel.

In C   int HspReadFormula(int *channel*, char **formula*);

In BASIC   Declare Function HspReadFormula Lib "hspcomm" (_
                       ByVal *channel* As Integer, _
                       ByRef *formula* As String _
) As Integer

                     channel A channel number.
                     formula Points *formula to formula string in the rx_buffer.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                     See Error Codes in Appendix C.

Remarks   This function will set the pointer (char **formula) to point to the ASCIIZ string
                     (an ASCII string terminated by a null byte) that is the formula for the specified
                     channel.

                     Note: This function does not copy the string to a destination string. The
                     formula string is located in the receive buffer and formula points to the
                     receive buffer location. The formula argument is a pointer to a string that
                     contains a valid formula. See the description of formulae in the section **HSP-
                     8 FORMULA SYNTAX** for details of allowed formats.

Function HspClearAllChannels

Description   Erases the formulae in all HSP-8 channels.

In C   int HspClearAllChannels(void)

In BASIC   Declare Function HspClearAllChannels Lib "hspcomm" (_
) As Integer

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                         See Error Codes in Appendix C.

Function HspClearChannel

Description   Erases the formula for the specified HSP-8 channels.

In C   int HspClearChannel(int *channel*);

In BASIC   Declare Function HspClearChannel Lib "hspcomm" (_
  ByVal *channel* As Integer _
) As Integer

channel A channel number.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   None

Function HspReadChannel

Description   Computes the value of the specified channel and returns the value.

In C   int HspReadChannel(int *channel*, float **value*);

In BASIC   Declare Function HspReadChannel Lib "hspcomm" (_
  ByVal *channel* As Integer, _
  ByRef *value* As Single _
) As Integer

*channel*A channel number.
value   The current value of the channel.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   None

Function HspSetCzero

Description   Sets the zero offset for the specified channel.

In C   int HspSetCzero(int *channel*, float **zero*);

In BASIC   Declare Function HspSetCzero Lib "hspcomm" (_
　ByVal *channel* As Integer, _
　ByRef *zero* As Single _
　) As Integer

*channel*A channel number.
zero　　The new zero offset value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The channel's zero offset value is added to the channel value when the
HspReadChannel function is used and when the channel is used in a formula
for another channel.

Function HspReadCzero

Description   Gets the zero offset for the specified channel.

In C   int HspReadCzero(int *channel*, float **zero*);

In BASIC   Declare Function HspReadCzero Lib "hspcomm" (_
  ByVal *channel* As Integer, _
  ByRef *zero* As Single _
) As Integer

*channel*A channel number.
zero   The current zero offset value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The channel's zero offset value is added to the channel value when the
HspReadChannel function is used and when the channel is used in a
formula for another channel.

All functions return the status as the function return value. When other values
must be returned the calling routine must supply a pointer to a variable which
will have the return value stored into it.

Function HspResetMM

Description   Resets all MIN/MAX/TIR elements used in channel formulae.

In C   int HspResetMM(void)

In BASIC   Declare Function HspResetMM Lib "hspcomm" (_
) As Integer

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                     See Error Codes in Appendix C.

Remarks   None

Function HspReadLvdt

Description   Gets the current value of the specified probe.

In C   int HspReadLvdt(int *probe*, float **value*);

In BASIC   Declare Function HspReadLvdt Lib "hspcomm" (_
  ByVal *probe* As Integer, _
  ByRef *value* As Single _
) As Integer

    probe   A probe number.
    value   The probe value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32. The *value* is determined by:
value = *fsv* * *amplifier value* + *offset*
Where the *amplifier value* ranges from -4.096 to +4.095.

Function HspSetTfsv

Description   Sets the full-scale value of the specified probe amplifier.

In C   int HspSetTfsv(int *probe*, float **fsv*)

In BASIC   Declare Function HspSetTfsv Lib "hspcomm" (_
                      ByVal *probe* As Integer, _
                      ByRef *fsv* As Single _
) As Integer

                     probe   A probe number.
                     fsv   The new full-scale value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                     See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
                     See **HspReadLvdt** for explanation of how this affects readings.

Function HspReadTfsv

Description   Gets the current full-scale value of the specified probe amplifier.

In C   int HspReadTfsv(int *probe*, float **fsv*)

In BASIC   Declare Function HspReadTfsv Lib "hspcomm" (_
                         ByVal *probe* As Integer, _
                         ByRef *fsv* As Single _
) As Integer

                       probe   An probe number.
                       fsv   The current full-scale value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                       See Error Codes in Appendix C.

Remarks   The probe argument must be in the range 1 to 32.
                       See **HspReadLvdt** for explanation of how this affects readings.

Function HspSetTzero

Description	Sets the zero offset of the specified probe amplifier.
In C	int HspSetTzero(int *probe*, float **zero*);

In BASIC	Declare Function HspSetTzero Lib "hspcomm" (_
	  ByVal *probe* As Integer, _
	  ByRef *zero* As Single _
	  ) As Integer
	
	probe A probe number.
	zero The new probe amplifier zero value.

Return Value	If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.
Remarks	The *probe* argument must be in the range 1 to 32. See **HspReadLvdt** for explanation of how this affects readings.

Function HspReadTzero

Description   Gets the current zero offset for the specified probe amplifier.

In C   int HspReadTzero(int *probe*, float **zero*);

In BASIC   Declare Function HspReadTzero Lib "hspcomm" (_
                         ByVal *probe* As Integer, _
                         ByRef *zero* As Single _
) As Integer

                       probe   A probe number.
                       zero   The current zero offset value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                       See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
                       See **HspReadLvdt** for explanation of how this affects readings.

Function HspReadAnalog

Description   Gets the specified analog input value.

In C   int HspReadAnalog(int *analog*, float **value*);

In BASIC   Declare Function HspReadAnalog Lib "hspcomm" (_
  ByVal *analog* As Integer, _
  ByRef *value* As Single _
) As Integer

analog An analog number.
value   The analog value.

Return Value If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   This function reads the current value of an analog input. The HSP-8 has no analog inputs. This function returns a value that reflects the state of the RTS line on the RS-232 connector.

Function HspSetAfsv

Description   Sets full-scale value for the specified analog input.

In C   int HspSetAfsv(int *analog*, float **fsv*)

In BASIC   Declare Function HspSetAfsv Lib "hspcomm" (_
  ByVal *analog* As Integer, _
  ByRef *fsv* As Single _
) As Integer

analog An analog number.
fsv   The analog full-scale value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   This function sets the full-scale value of an analog input.
See Remarks for **HspReadAnalog**

Function HspReadAfsv

Description   This function gets the full scale value of an analog input.

In C   int HspReadAfsv(int *analog*, float **fsv*);

In BASIC   Declare Function HspReadAfsv Lib "hspcomm" (_
  ByVal *analog* As Integer, _
  ByRef *fsv* As Single _
) As Integer

analog An analog number.
fsv   The analog full scale value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   This function reads the full scale value of an analog input.
See Remarks for **HspReadAnalog**

Function HspSetAzero

Description   Sets the zero offset for the specified analog input.

In C   int HspSetAzero(int *analog*, float **zero*);

In BASIC   Declare Function HspSetAzero Lib "hspcomm" (_
  ByVal *analog* As Integer, _
  ByRef *zero* As Single _
) As Integer

analog An analog number.
zero   Pointer to the analog zero offset value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The zero offset is added to the analog input value when the read analog
function is used or when the analog input is used in a formula.
See Remarks for **HspReadAnalog**

Function HspReadAzero

Description   Gets the zero offset for the specified analog input.

In C   int HspReadAzero(int *analog*, float **zero*);

In BASIC   Declare Function HspReadAzero Lib "hspcomm" (_
    ByVal *analog* As Integer, _
    ByRef *zero* As Single _
) As Integer

analog An analog number.
zero   Pointer to the analog zero offset value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   See Remarks for **HspReadAnalog**

Function HspSetLvdtGain

Description   Sets probe gain resistors on the specified probe amplifier.

In C   int HspSetLvdtGain(int *probe*, unsigned char *Rf*, unsigned char *Ri*);

In BASIC   Declare Function HspSetLvdtGain Lib "hspcomm" (_
                       ByVal *probe* As Integer, _
                       ByVal *Rf* As Byte,_
                       ByVal *Ri* As Byte_
) As Integer

                     probe   A probe number.
                     Rf   Feedback resistor code
                     Ri   Input resistor code

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                     See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
                     The feedback resistor code, *Rf*, must be in the range 0 to 255.
                     The input resistor code, *Ri*, must be in the range 0 to 255.

                     See Section on Adjusting Gain.

Function HspReadLvdtGain

Description   Gets probe gain resistors on the specified probe amplifier.

In C   int HspReadLvdtGain(int *probe*, unsigned char *Rf, unsigned char *Ri);

In BASIC   Declare Function HspReadLvdtGain Lib "hspcomm" (_
                         ByVal *probe* As Integer, _
                         ByRef *Rf* As Byte,_
                         ByRef *Ri* As Byte_
) As Integer

                       probe   A probe number.
                       Rf   Feedback resistor code.
                       Ri   Input resistor code.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                       See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
                       The feedback resistor code, *Rf*, will be in the range 0 to 255.
                       The input resistor code, *Ri*, will be in the range 0 to 255.

                       See Section on Adjusting Gain.

Function HspSetLvdtFine

Description   Sets fine gain trim on the specified probe amplifier.

In C   int HspSetLvdtFine(int probe, short int *fine*);

In BASIC   Declare HspSetLvdtFine Lib "hspcomm" (_
  ByVal *probe* As Integer, _
  ByVal *fine* As Integer_
) As Integer

probe   A probe number.
fine   The gain fine adjust value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
The *fine* argument must be in the range -100 to +100.
See Section on Adjusting Gain.

Function HspReadLvdtFine

Description   Gets fine gain trim on the specified probe amplifier.

In C   int HspReadLvdtFine(int *probe*, short int *fine);

In BASIC   Declare Function HspReadLvdtFine Lib "hspcomm" (_
  ByVal *probe* As Integer, _
  ByRef *fine* As Integer_
) As Integer

probe   A probe number.
fine   The current fine gain trim value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
The returned value of *fine* will be in the range -100 to +100.

See Section on Adjusting Gain.

Function HspSetLvdtNull

Description   Sets null trim on the specified probe amplifier.

In C   int HspSetLvdtNull(int *probe*, short int *offset*);

In BASIC   Declare Function HspSetLvdtNull Lib "hspcomm" (_
  ByVal *probe* As Integer, _
  ByVal *offset* As Integer_
) As Integer

     probe   A probe number.
     offset   The new offset zero trim value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
The *offset* argument must be in the range -100 and +100.

See Section on Adjusting Gain.

Function HspReadLvdtNull

Description   Gets null trim on the specified probe amplifier.

In C   int HspReadLvdtNull(int *probe*, short int **offset*);

In BASIC   Declare Function HspReadLvdtNull Lib "hspcomm" (_
  ByVal *probe* As Integer, _
  ByRef *offset* As Integer, _
) As Integer

  probe   A probe number.
  offset   The current offset zero trim value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The *probe* argument must be in the range 1 to 32.
The *offset* argument must be in the range -100 and +100.

See Section on Adjusting Gain.

Function HspSaveCalibration

Description   Commits the current calibration to non-volatile memory.

In C   HspSaveCalibration(short int *savestatus*);

In BASIC   Declare Function SaveCalibration Lib "hspcomm" (_
  ByRef *savestatus* As Integer_
) As Integer

savestatus   The status of the save request is stored here.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   Save calibration to eeprom.

savestatus = 0 means saving of calibration data has started
savestatus = any other value means the firmware was unable to start the
save operation (most likely a previous save has not yet completed)

Function HspGetSaveStatus

Description   Checks for non-volatile memory write complete.

In C   int HspGetSaveStatus(short int *savecomplete*, short int *saveerror*);

In BASIC   Declare Function HspGetSaveStatus Lib "hspcomm" (_
                          ByRef *savecomplete* As Integer, _
                          ByRef *saveerror* As Integer_
) As Integer

                        savecomplete   See Remarks.
                        saveerror   See Remarks.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
                        See Error Codes in Appendix C.

Remarks   *savecomplete* = 1 means saving is done and *saveerror* = 0 means saving
                        was successful
                         savecomplete = 0 means saving is in progress and *saveerror* has no
                        meaning

Function HspSetScanTime

Description   Sets scanning period for MAX, MIN and TIR functions.

In C   int HspSetScanTime(short int *time*);

In BASIC   Declare Function HspSetScanTime Lib "hspcomm" (_
  ByVal *time* As Integer _
) As Integer

         time   New scanning period value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   The argument time is in units of tenths of milliseconds.

Function HspGetScanTime

Description   Gets the scanning period for MAX, MIN and TIR functions.

In C   int HspGetScanTime(short int *time*);

In BASIC   Declare Function HspGetScanTime Lib "hspcomm" (_
  ByRef *time* As Integer _
) As Integer

  time   The current scanning period value.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value.
See Error Codes in Appendix C.

Remarks   The argument *time* is in units of tenths of milliseconds. Example: A value of
100 means 10.0 milliseconds.

Function HspGetScanFlag

Description   Gets the current state of the HSP scan flag.

In C   int HspGetScanFlag(short int *flag*);

In BASIC   Declare Function HspGetScanFlag Lib "hspcomm" (_
  ByRef *flag* As Integer _
) As Integer

              flag   Read the current scan flag setting.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   Read the current scan flag setting.

Function HspReadDirect

Description	Gets probe amplifier values as signed integers.
In C	int HspReadDirect(int *count*, unsigned char **list*, short int ***values*);

In BASIC	Declare Function HspReadDirect Lib "hspcomm" (_
	  ByVal *count* As Integer, _
	  ByRef *list* As Byte, _
	  ByRef *values* As Integer _
	  ) As Integer
	
	count Number of probe's to be read.
	list Pointer to a list of probe numbers.
	values Pointer to an integer pointer.
	The calling routine passes the address of a variable where the pointer to the table of values is to be stored.

Return Value	If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.
Remarks	The probe numbers are 8 bit integers starting at "0" through "7" for the first HSP-8, "8" through "15" for the second HSP-8, etc. The foot switch input is "96". Each of the returned integers or values will be in the range of -8192 to +8191. See Remarks on **ReadDirectCopy** function.

Function HspReadDirectCopy

Description   Reads the requested amplifiers and copies the results to the buffer provided by the calling routine.

In C   int HspReadDirectCopy(int *count*, unsigned char **list*, short int **values*);

In BASIC   Declare Function HspReadDirectCopy Lib "hspcomm" (_
  ByVal *count* As Integer, _
  ByRef *list* As Byte, _
  ByRef *values* As Integer _
) As Integer

  count   Number of probe's to be read.
  list   Pointer to the list of probe numbers.
  values Pointer to buffer where values are to be stored.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   The probe numbers are 8 bit integers starting at "0" through "7" for the first HSP-8, "8" through "15" for the second HSP-8, etc. The foot switch input is "96".
Each of the returned integers or values will be in the range of -8192 to +8191.
The buffer must be large enough to hold count 16 bit integers.
The **HspReadDirectCopy** and **HspReadDirect** functions do the same thing, except for how the results are transferred back to the calling program. The **HspReadDirectCopy** function is useful when using some BASIC compilers which provide no argument passing option which is equivalent to the C int **values argument.

Function HspSetAdcAvg

Description   Sets burst average length.

In C   int HspSetAdcAvg(short int *NumberToAvg*);

In BASIC   Declare Function HspSetAdcAvg Lib "hspcomm" (_
  ByVal *NumberToAvg* As Integer _
) As Integer

NumberToAvg New burst average length.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   Each amplifier or analog request will be read NumberToAvg times. The returned value will be the average of these readings. Each of the readings is very closely spaced in time (about 5us per reading).

At startup the burst average is set to 4.

Function HspSetRunAvg

Description   Sets running average length.

In C   int HspSetRunAvg(short int *NumberToAvg*);

In BASIC   Declare Function HspSetRunAvg Lib "hspcomm" (_
  ByVal *NumberToAvg* As Integer _
) As Integer

   NumberToAvg   New running average length.

Return Value   If successful, the function returns 0. Otherwise, it returns a nonzero value. See Error Codes in Appendix C.

Remarks   The running average will accumulate one reading per scan period up to *NumberToAvg*, at which time it will discard the oldest value. When the value of a channel is requested, the average of the accumulated values for that channel will be returned. The running average only effects values from **HspReadChannel**. It has no effect on **HspReadDirect** or **HspReadLvdt**.

   At startup the running average is set to 1.

Adjusting Probe Amplifier Gain

Each probe amplifier has two software adjustable resistors that together set the voltage gain of the probe amplifier. The two resistors are referred to as the input resistor (Ri) and the feedback resistor (Rf). The probe amplifier gain is determined by the following formula:

$$Gain = 1 + Rf / Ri$$

Although both resistors are adjustable, it is recommended that the input resistor (Ri) be set and left at the value of 50. This allows the gain to be varied from a minimum of 1 (Rf=0) to a maximum of a little more than 6 (Rf=255). This range will accommodate most probes that are used with the probe amplifier. If high output probes are used in combination with a high Gain setting, the detector will overload and cause signal polarity reversal. This will not cause any permanent damage, but the resulting readings will be erroneous.

The functions **HspSetLvdtGain** and **HspReadLvdtGain** are used to manipulate the Rf and Ri values on a per amplifier basis. The functions **HspSetLvdtFine** and **HspReadLvdtFine** manipulate a fine gain adjust factor, which is useful if it is desired to set a gain between the steps allowed by the setting of Rf.

The **HspSetLvdtNull** and **HspReadLvdtNull** are useful for removing any residual reading that exists with the probe amplifier inputs open circuit.

Once the calibration is satisfactory, it can be made the power-up default by using the **HspSaveCalibration** function. This function will commit the calibration values to a non-volatile memory. Since the **HspSaveCalibration** function has the potential for taking a long* time to complete, the **HspGetSaveStatus** function should be used to determine when the save has completed before other functions are used. The **HspSaveStatus** function will also report if the save was completed successfully.

* A long time is only a few seconds, but that is long relative to the other functions that only take a few *thousandths* of a second.

Appendix A: HSP-8 Formula Syntax

When a channel is read, its value is computed by a formula supplied to the HSP-8 by the host software. The formulae are supplied to the HSP-8 in the form of an ASCII string terminated by a null byte. Formulae are constructed in much the same way as they are in assignment statements in programming languages such as BASIC.

Formulae consist of combinations of functions, operators, constants, input terms and channel terms.

Peak Hold Functions

MAX returns the largest value it sees as an argument while scanning.
MIN returns the smallest value it sees as an argument while scanning.
TIR returns MAX-MIN.
The MAX, MIN and TIR functions continue to be scanned although the channel that uses them is not being read by the host software. They will 'freeze' at their current value if scanning is disabled. The scanning is controlled by the functions **HspStartScan** and **HspStopScan**. The period of scanning is set by the **HspSetScanTime** function. They are reset by the **HspResetMM** function.

Mathematical Functions

ABS returns the absolute value of the argument.
ACOS returns the arc-cosine of the argument. Result in radians.
ASIN returns the arc-sine of the argument. Result in radians.
ATAN returns the arctangent of the argument. Result in radians.
COS returns the cosine of the argument. Argument in radians.
GOF returns the greatest (most positive) of the argument list.
GOR returns the greatest (most positive) of the range set by the argument list.
LOF returns the least (most negative) of the argument list.
LOR returns the least (most negative) of the range set by the argument list.
SIN returns the sine of the argument. Argument in radians.
SQRT returns the square root of the argument.
SQR returns the argument multiplied by itself.
TAN returns the tangent of the argument. Argument in radians.

Miscellaneous Functions
PI2 has the value 3.141592654/2.0
PI has the value 3.141592654
RAD returns the argument converted from degrees to radians.
DEG returns the argument converted from radians to degrees.
(and **)** are allowed to impose computation order.

Mathematical Operators
^ exponentiation
* multiplication.
/ division. (division by zero returns zero)
+ addition.
- subtraction.

Constants
Constants in formulae are specified as ASCII strings. Scientific notation is not allowed for constants, (i.e., '.125' is allowed, '1.25E-01' is not allowed).

Input Terms

Tn returns the value of transducer number n. The number n must be in the range 1 to 32. The HSP-8 will read the transducer, multiply by the transducer full scale value and add the transducer offset. Each transducer has an individual full scale value and offset.

Channel Terms

Cn returns the value of channel number n. The number n must be in the range 1 to 96. The HSP-8 will compute the value of channel nn, multiply by the channel full scale value and add the channel offset. Each channel has an individual full scale value and offset. Circular references to channels are not allowed and will cause the HSP-8 to return an error code.

Example Formulae

T1
T1+T2
MAX(T2-T1)
(T1+T2+T3)/3
1.0034*(T1+T2)
(MAX(T1)+MIN(T1))/2
TIR(T3) same as MAX(T3)-MIN(T3)
GOF(T1,T2,T3) returns the greatest of T1, T2 and T3.
LOF(T1,T2,T3,T4) returns the least of T1, T2, T3 and T4.
GOR(T1,T8) returns the greatest of T1,T2,T3,T4,T5,T6,T7 and T8.
LOR(C9,C13) returns the least of C9,C10,C11,C12 and C13.

Appendix B: Startup Settings

Immediately after turning on the HSP-8, before any HSP-8 functions are executed, the following settings will be in effect:

All 96 channel scale factors are set to 1.
All 96 channel zero offsets are set to zero.
All 32 transducer full scale values are set to .02.
All 32 transducer zero offsets are set to zero.
The burst average is set to 4.
The running average is set to 1.
The floating point input and output modes are both set to the native (IEEE) mode.

Appendix C: Error Codes

Communication Error

-1 The HSP-8 is not responding to communication from the host PC. Is the HSP-8 plugged in and powered?

Command Error

1 An invalid command or parameter was entered.

2 - 9: Reserved.

Channel Definition Errors

When defining channels, error codes **10-22** may be returned. They may be used to help determine the part of the formula that is causing the problem.

@BULLET = **10** An invalid channel number was entered.

@BULLET = **11** Internal error.

@BULLET = **12** Invalid opcode mnemonic.
A built-in function name has been improperly typed in. If "SINE(T4)" is entered instead of "SIN(T4)" or "T1+S3" instead of "T1+T3".

@BULLET = **13** Not enough operands for operator.
If "T1+" is entered the HSP-8 cannot determine what value to add to T1. If "SIN()" is entered the HSP-8 cannot determine what value to use as the argument of the sine function.

@BULLET = **14** Node table full.
The HSP-8 reduces the entered formulae into a form that can be quickly computed when required. The reduced formulae are stored in a table that has a predetermined size. When the table is filled this error is returned. If this error occurs, the number or the complexity of the entered formulae must be reduced.
Determining the number of node entries in the node table that a particular formula consumes is difficult due to certain optimizations which the HSP-8 applies during the formula reduction process.

The following guidelines will allow determining the maximum number of nodes that a given formula will consume.

1. Each constant consumes one node. The formula ". 0023" consumes one node.

2. Each function use consumes one node for the function. Additional nodes will be consumed for the function's arguments. The formula "SIN(.0023)" consumes one node for the SIN function plus one node (Rule 1) for the constant .0023, for a total of 2 nodes. The TIR function is a special case which consumes 2 nodes just for the function plus the additional nodes for the function arguments. The formula "TIR(.0023)" consumes 2 nodes for the TIR function and 1 node for the constant, for a total of 3 nodes.

3. Arithmetic operators (+ - * /) consume one node. The formula "1 + 2 + 3" consumes 1 node for each of the "+" operators plus 1 node for each of the constants (Rule 1), for a total of 5 nodes.

4. Channel references consume one node. The formula "C1" consumes 1 node.

5. Transducers and analog inputs consume one node. The formula "T1" consumes 1 node. The formula "T1 + T2" consumes 1 node for each of the transducers plus 1 node for the "+" operator (Rule 3), for a total of 3 nodes. The first appearance of a transducer in a formula consumes 1 node however subsequent appearances of the same transducer in any formula do not consume additional nodes.
 Channel 1 is "T1+T2". This uses 3 nodes.
 Channel 2 is "T1+T3". This uses 2 nodes. T1 has already been allocated
a node by the channel 1 formula.

The total number of nodes available is 400.
The HSP-8 cannot conserve nodes by recognising common subexpressions
in formulae. Node table space can be conserved by defining a channel
with the subexpression and then referencing that channel in the formulae
which need the subexpression value.

Channel 1 is "T5 - (T1 + T2 + T3 + T4)". (9 nodes)
Channel 2 is "T6 - (T1 + T2 + T3 + T4)". (5 nodes)
Channel 3 is "T7 - (T1 + T2 + T3 + T4)". (5 nodes)
Channel 4 is "T8 - (T1 + T2 + T3 + T4)". (5 nodes)
For a total of 9+5+5+5 or 24 nodes.

Alternately define channel 50 with the subexpression and use channel
50 in place of the subexpression:
 Channel 50 is "T1 + T2 + T3 + T4". (7 nodes)
 Channel 1 is "T5 - C50. (3 nodes)
 Channel 2 is "T6 - C50. (3 nodes)
 Channel 3 is "T7 - C50. (3 nodes)
 Channel 4 is "T8 - C50. (3 nodes)
For a total of 7+3+3+3 or 16 nodes.

@BULLET = 15 Bad transducer or analog input number.
If a formula contains "T175" or "A32". The highest transducer number
allowed is "T32". The highest analog input number allowed is "A1".
Note: Although the HSP-8 accepts transducer numbers up to 32 the slave
HSP-8's must be present for these to produce any meaningful measurements.

@BULLET = 16 Too many operands for operator.
If "T1 + T2 T3" is entered, the HSP-8 cannot determine what to do
with the T3 part of the formula.

@BULLET = 17 Bad numeric value in expression.
When entering constants in a formula "scientific notation" is not
allowed. In some computer languages the value .0015 may be entered
as "1.5E-3". The HSP-8 requires that the value be entered as ".0015".

@BULLET = 18 Bad token. An invalid symbol was entered.

@BULLET = 19 Formula too complex to parse.
This error occurs when a formula has more levels of parenthesis than
the HSP-8 can handle

@BULLET = 20 Recursive channel definition.
This error occurs when channels reference each other in a circular
fashion.
 Channel 1 has the formula "T1+T2-C2".
 Channel 2 has the formula "T3+C1".
To determine the value of channel 1 the HSP-8 must first determine

the value of channel 2, which is dependent on the value of channel 1.

@BULLET = 21 No memory left to allocate.
Besides storing the formulae in a reduced form (as described under error 14), the original text of the formula is also saved within the HSP-8 memory. The formula text is stored in a memory pool along with other items which are necessary during formula entry. Should this pool of memory become filled this error is issued.

@BULLET = 22 General formula error.
The formula scanner in the HSP-8 will issue this error when the formula is bad and no other formula error codes apply.