TABLE OF CONTENTS
PROGRAMMING GUIDE

SECTION 3.0
SIGLAB PROGRAMMER'S REFERENCE

SigLab Interface Overview

The first part of this section describes the function calls associated with the **siglab.dll**. This file is the MEX interface which communicates with SigLab hardware in the MATLAB environment. The second part describes the various Long Record Capture file utilities provided with **vcap**.

The general syntax for the siglab function is:

```
[out1, out2, ...]=siglab(Action,in1, in2, in3, ....)
```

where:

Action	A string determining the command (e.g. 'IOinit').
`in1 in2`, etc.	Input arguments which can be strings, scalars, or vectors. Note: All input vectors may be either row or column vectors.
`out1 out2`, etc.	The returns from the function call.

The various actions performed by the SigLab DLL are listed alphabetically below, followed by a functional group listing. The full definitions follow in alphabetical order. At the end of the section transient capture file access functions, which are not contained within siglab.dll, are described.

CapStatus	Retrieve transient capture status
Capture	Initiate transient capture
Compute	Specify functions to compute/average
DataAbort	Cancel data request and free buffer

DataGet	Get requested data
DataRdy	Check for availability of previously requested data
DataReq	Request data from SigLab
Debug	Set Debug state of dll/ perform various debug operations
Delayms	Pause execution for specified time
Event	Start or stop a processing operation (e.g. AvgStart)
Execute	Execute a program to process a file
Get	Determine if Bias Source is available
GetStat(I)	Request overload Status
Help	Invoke Windows Help system
InpGain	Set input channel gain, offset and coupling
InpSet	Set input acquisition parameters (e.g. Bandwidth)
InStatus	Return overload and other status information
Ioinit	Initialize dll and download microcode to SigLabs
Lattice	Invoke recursive least squares lattice/predictor
OutBurst	Set output channel burst mode(for Random, Arb or Chirp)
OutLevel	Set output channel gain and offset
OutPhase	Set phase between sine outputs
OutPulse	Set output channel periodic function(Sqr, Tri, Saw or Impulse)
OutSine	Set output channel sinusoidal output

Playback	Control playback from transient capture buffer
Process	Set averaging and window parameters
RawCommand	Low-level command
RPMreq	Request Speed information
SendArb	Send data block to arbitrary output data buffer
SendCal	Calibration support functions
SendInt	Send data block to transient capture buffer
SetWindow	Set user defined processing window
StateSpace	Set up state space controller
StatusCallBack	Define callback function for overloads
Trigger	Set input triggering parameters

SigLab.dll actions grouped by function

Input Setup Functions

Compute	Specify functions to compute/average
Get	Determine if Bias Source is available
InpGain	Set input channel gain, offset and coupling
InpSet	Set input acquisition parameters (Bandwidth)
Process	Set averaging and window parameters
SetWindow	Set user defined processing window
Trigger	Set input triggering parameters

Input Data Request and Status Functions

Event	Start or stop a processing operation (e.g.

	AvgStart)
DataAbort	Cancel data request and free buffer
DataGet	Get requested data
DataRdy	Check for availability of previously requested data
DataReq	Request data from SigLab
GetStat(I)	Request overload Status
InStatus	Return overload and other status information
RPMreq	Request Speed information
StatusCallBack	Define callback function for overloads

Output Generator Setup Functions

OutBurst	Set output channel burst mode (for Random,Arb or Chirp)
OutLevel	Set output channel gain and offset
OutPhase	Set phase between sine outputs
OutPulse	Set output channel periodic function(Sqr, Tri, Saw, or Impulse)
OutSine	Set output channel sinusoidal output
SendArb	Send data block to arbitrary output data buffer

Miscellaneous Functions

Debug	Set Debug state of dll/ perform various debug operations
Delayms	Pause execution for specified time
Execute	Execute a program to process a file

IOinit	Initialize dll and download microcode to SigLabs
Lattice	Invoke recursive least squares lattice/predictor
RawCommand	Low level command
SendCal	Calibration support functions

Transient Capture and Playback Functions

CapStatus	Retrieve transient capture status
Capture	Initiate transient capture
Playback	Control playback from transient capture buffer
SendInt	Send data block to transient capture buffer

SigLab.dll Actions - Detailed Descriptions

Transient Capture Status

'CapStatus', Operation

Syntax:

```
siglab('CapStatus',Operation);
```

Purpose: Retrieves information regarding transient capture status.

Operation is one of the following:

'done'	Returns 1 if transient capture is complete, 0 otherwise. Clears done flag such that if called twice in a row, the 2nd time will return 0.
'donecallback',funcname ,funcarg	Specifies a Matlab function to handle completion status callback. Must be followed by two arguments: the first is the name of the function, the second is the first argument passed to the function and may either be an integer of a string.. Note that this call need be made only once, preferably before the first capture has been

started. To cancel the callback, pass a null string (') as the first argument.

For example:

siglab('CapStatus','donecallback','vcap','getdonestatus');

would result in the call vcap('getdonestatus',1,1) being made when the capture is complete.

'wait' Waits until new status info is received or until 1 second passes. Useful only with the following parameters - not with 'done', which normally will not be available within a second.

'bufsize' Returns size of transient capture buffer in bytes after a siglab('playback','getsize') call.

All of the following status items are only available after a siglab('playback','getstatus') call

'discon' Returns 1 if data in tc buffer is discontinuous

'frames' Returns number of frames in buffer

'samps' Returns number of samples in buffer

'ovld' Returns overload code

'cursamps' Returns position in samples for current frame being played back

Long Record Support

'Capture'

Syntax:

```
siglab('Capture',Chanb,Framecount);
```

Purpose: Initiates transient capture, using the acquisition setup most recently specified using InpSet, InpGain, and other setup commands.

Input Arguments:

Chanb A vector of channels to be captured.

Framecount	Number of frames to acquire in transient capture buffer.

Functions to Compute

'Compute'

Syntax:

```
siglab('Compute',Aspecs,Response,Reference);
```

Purpose: Defines which auto and cross spectra are to be computed during the averaging process. No cross channel averages will be done if the Response and Reference parameters are missing. If the Response vector is included, the Reference parameter must also be supplied and size (Response) must equal size (Reference). If time-domain averaging has been selected then the Aspecs vector selects which channels should be time-domain averaged. Note that in multibox systems, only Reference channels in the first box are permitted.

Input Arguments:

Aspecs	A vector of chans specifying which ASPECs to average.
Response	A vector of chans specifying the response channel for each CSPEC to be averaged.
Reference	A vector of chans specifying the reference channel for each CSPEC to be averaged.
Returns:	0 if a valid set of reference and response channels were specified, -1 otherwise.

Abort Request

'DataAbort'

Syntax:

```
siglab('DataAbort',ReqID);
```

Purpose: To cancel a request for data that has already been made.

Note that each ReqID returned by a 'DataReq' call must be closed out with either a 'DataGet' or 'DataAbort' call with that ReqID.

Input Arguments:

ReqID	Specify the data set that is no longer desired. This is the value returned by 'DataReq'

Returns:

Nothing.

Get Data

'DataGet'

Syntax:

```
[Dest Ovld Header] =
siglab('DataGet',ReqID,'OptionalParameters');
```

Purpose: To retrieve measurement data once it is available to be ready. This call must be preceded by a DataReq and should be preceded by a DataRdy (or performed in callback)

Input Arguments:

ReqID	Specify the data of interest with the tag returned from DataReq.
'Scale',SC	The function associated with the nth element of the Chan vector specified in the DataReq command will be multiplied by scale factor SC(n). If SC is shorter than n elements or if this parameter is not specified, then the scale factor will default to 1.0 .

Returns:

Dest	Scaled measurement results; # of columns = size(Chan); # of rows = NOE.
Ovld	Same form as Ovld returned from SigLab ('InStatus'). A channel is included if an overload occurs during any frame of the average.
Header	The optional header parameter has only rather obscure utility. It can be used to verify that the associated function data came from a particular input frame, by examining the seqNum field, or to

check the number of averages with which an instantaneous time or aspec frame is associated with (the numavgs field)

The Header information is returned in an array of 3 double precision values for each channel or function returned in Dest. These three values, when converted to 32 bit integers, contain the following bit fields:

First 32-bit word : Header(1,x) :

unsigned int numel:16; /* number of elements returned */

unsigned int func:5; /* function code :

0 Instantaneous Time
1 Averaged Time
2 Instantaneous Auto Spectrum
3 Average AutoSpectrum
4 Cross Spectrum
5 Transfer Function
6 Impulse Response
7 Auto Correlation
8 Cross Correlation
9 Coherence
10 Instantaneous Zoom Time
11 Averaged Zoom Time
12 FFT
13 Reserved
14 Decimated Time*/
unsigned int boxnum:3; /* Siglab Box # (0..) */
unsigned int chan:5; /* channel #-1(0..) */
unsigned int Ovld:1; /* overloaded data in function */
unsigned int Ovldresp:1; /* overloaded data in response */
unsigned int Ovldref:1; /* overloaded data in reference */
Second 32-bit word : Header(2,x) :
unsigned int reqID:5; /* request ID */
unsigned int seqNum:5; /* acquisition frame sequence number */

unsigned int refchan:5; /* ref channel for cross functions */

unsigned int inDiscon:1; /* discontinuous data */

unsigned int numavgs:16; /* number of averages in data */

Third 32-bit word : Header(3,x) :

unsigned long hdrlong; /*sample offset or Speed */

};

Check for Data

'DataRdy'

Syntax:

```
Rdy = siglab('DataRdy',ReqID);
```

Purpose: To inquire about the existance of previously requested measurement data. It is suggested that DataRdy be used even when DataReq is called with the “wait” argument in order to obtain the status information. This function is not required if a callback was specified in the DataReq.

Input Arguments:

ReqID	Specify the data of interest with the tag returned from the DataReq function.

Returns:

Rdy	0 if data not ready; -1 if data was never requested. -2 indicates a SCSI bus error. Otherwise Rdy = # of averages completed. (or 1 when not averaging)

Data Request

'DataReq'

Syntax:

```
ReqID = siglab('DataReq',NOE,Chan,'OptionalParameters');
```

Purpose: Request measurement data from SigLab system with a specified format. A ReqID tag is returned to facilitate checking on when the data is available and actually obtaining it. Any successful request must be closed out by a successful DataGet or DataAbort.

Input Arguments:

NOE	# of elements requested (per channel).
Chan	A channel number or a vector of several channels requested (The channels must be listed in increasing numerical order).
'Ref',Rch	Reference channels; size (Rch) must equal size(Chan) (Default: if Chan is odd, Rch = Chan+1, otherwise Chan-1).
'AspecI'	Function requested. Choices are: TimeI (default), TimeA, AspecI, AspecA, Cspec, Xfer, Impulse, Acor, Ccor, Coh.
'First',#	Index of the first point desired (Default = 0).
'Wait'	DataReq waits internally until requested data is available. This is a dangerous option as the program will hang here forever if the data is not available.
'NoWait'	DataReq returns immediately after making request (default).
'Immed'	If this parameter is present then the data will be returned (immediately) even if the data has been displayed and/or processed.

'Callback',funcname ,funcarg	Specifies Matlab function which will be handle the callback when data becomes available. If this option is used, then the callback routine will perform the function of siglab('datardy'). This parameter must be followed by two parameters: funcname, a string containing the name of the matlab function to be called, and funcarg, which is either a string or a scalar. When the data associated with the request ID returned by 'DataReq' is ready, a callback will be made to funcname, as follows: funcname(funcarg,resultcode,ReqID) where resultcode is the value normally returned by 'DataRdy', i.e. a positive integer representing the number of averages if successful, or a negative value if an error occured. ReqID is the tag returned by this 'DataReq' call. This callback routine is expected to call either SigLab('DataGet',...) or SigLab('DataAbort',...) to complete the transaction.

Returns:

ReqID	Used to identify this data request with DataRdy, DataGet, DataAbort, and the callback function. If ReqID is < 0, then the request failed.

Debug Mode

'Debug'

Syntax:

```
siglab('Debug',DebugMode);
```

Purpose: Selects the siglab debugging mode.

Input Arguments:

Debug Mode

1:Disable SCSI communication to all SigLabs.

2:Echo all SigLab DLL calls to command window.

4 :Echo a hex representation of all SigLabSCSI commands.

8 : Enable certain debug messages

16 : Download to IndyBox

32 : Download to main boxes

DebugMode may also be the sum of any of these values. For example if DebugMode=7, then all SigLab commands will be echoed to the MATLAB command window in both ASCII and hex forms, but the commands will not actually be sent to the SigLab. Negative values perform various, obscure debugging functions as noted below.

Special (and Obscure!) Debug codes for Siglab('Debug',...

-1 Print outstanding requests

-2 Print all request slots

-3 Kill status request

-4 Unused

-5 Unused

-6 send SCSI Inquiry to scsi id (last if -1); prt if xtra arg
[serial number] = siglab('debug',-6,scsi id {,print}
-7 Turn wait mode OFF
-8 Turn wait mode ON
-9 Scan all SCSI targets and print # of SigLabs found
-10 Turn OFF low level DBmex SCSI debug messages
-11 Turn ON DBmex debug messages
-12 Perform Low Level Abort of all scsi transactions (SRBs)
-13 Print a bunch of internal scsi stuff (DBprint + following)
-14 Print memory info and ...
-15 Print srb list
-16 Set high address for download checksum
siglab('debug',-16, address);
-17 Set low address for download checksum
siglab('debug',-17, address);
-18 Return memory info
[sysheap dwmaxfree dwfree] = siglab('debug',-18);

-19 Print version info for all SigLab targets
-20 Return number of in / out chans for specific id
[serialnum inchan outchan modelcode] = siglab('debug',-20,scsi id);
-21 Send system date to box
siglab('debug',-21,scsi id);
-22 Reserved - do not use
-23 Return number of free SRBs
SRBs = = siglab('debug',-23);
-24 Return boot date and version strings[isValid dateString verString modelcode] = siglab('debug',-24,scsi id);
-25 Return code date and version strings
[isValid dateString verString modelcode] = siglab('debug',-25,scsi id);
-26 Get adaptor info
[isASPIdos ASPImgrName HostAdaptorName] = siglab('debug',-26);
-27 Return dll version
[dateString dllverstring] = siglab('debug',-27);
-28 Send Test WriteBuffer command (optional arg is bytecount)
-29 Enable SCSI aborts
-30 Disable SCSI aborts
-31 Write a 32bit word to siglab memory for last box used
siglab('debug',-31,address,value);
-32 Send SCSI BusDevicereset (default target is last used)
siglab('debug',-32,scsi id);
-33 Set host adaptor number(HA) for multiple adaptor tests
siglab('debug',-33,host adaptor #);
-34 Send SCSI tstunitrdy for (target,lun)
[status completionStatus targetStatus hostStatus] = siglab('debug',-34,scsi id,lun);
-35 print last error for target,lun (target = -1 = last target)
siglab('debug',-35,scsi id);
-36 Determine amount of DOS real mode memory available
dosheapsize = siglab('debug',-36); % win 3.1 only
-37 return siglab memory and status
[dramKb sramKb isrunning] = siglab('debug',-37,scsi id)
-38 Send SCSI change definition command
-39 Print last errors for all targets and luns
-40 Set Receive timeout limit in seconds
siglab('debug',-40,Limit_In_Seconds);
-41 Set Send timeout limit in seconds
siglab('debug',-40,Limit_In_Seconds);
-42 Return 1 if aborts are enabled
-43 Return and print download checksum

-44 Select Host adapter
-45 Return Host adapter number and total number of adapters
[HA TotalNumAdapters] = siglab('debug',-45)
-46 Set default SCSI target ID
-47 Read SigLab memory (1st word only is returned in result)
result = siglab('debug',-47,address,Numwords)
-48 Read SCSI target Inquiry string
[isScsi inqstring] = siglab('debug',-48,target);
-49 Set max download transfer size
-50 Print status of last Receive operation on specified LUN
-51 Enable/Disable boot patch for v1.02 proms
-52 Set segment size in bytes for Receive operations
-53 Set NoPosting
-54 Spawn rescan.exe to load SCSI drivers
-55
-56 Set DBnegREQ
-57 Set NoPosting
-58 Return latest speed value, or -1
-59 Unused
-60 Unlock mex file
-61 Lock mex file
-62 Set Stepped Sine Mode
siglab('debug',-75,StartFreq,DeltaFreq,NumSteps)
-63 Set DBallowMix22aAnd22
-64 Get Number of Siglabs and IndyState
[Numboxes IndyState] = siglab('debug',-77)
-65 Get Number of bytes free on disk
Numbytes = siglab('debug',-78,path)
-66 Call lyzerAtExit
-67 Call TestReceive for LUN 0
result = siglab('debug',-80,DelayticsX10,NumRcvReqs);
-68 Call TestReceive for LUN 1
-69 Call TestReceive for LUN 2
-70 Call TestReceive for LUN 3
-71 Call TestReceive for LUN 4
-72 Call TestReceive for LUN 5
-73 Call TestReceive for LUN 6
-74 Call TestReceive for LUN 7
-75 Unused
-76 Enable Status Requests
-77 Unused
-78 Unused
-79 Set DBinterBoxWait
-80 Set IndyState (0 = none, 1 = Main, 2 = last)
-81 Set EnableAutoInit = 1

-82 Request new status on specified box
-83 Print status request information
-84 Enable DBmsgs
-85 Disable Dbmsgs
-86 Suppress status request transfers

Execution Delay

'DELAYms'

Syntax:

siglab('DELAYms', Dly);

Purpose: Pauses execution for Dly milliseconds

Input Arguments:

Dly	Time delay in milliseconds (Dly from 1 to 1e6)

Process Control

'Event'

Syntax:

```
siglab('Event',ChanB,'Option');
```

Purpose: Start or stop a processing operation inside SigLab.

Input Arguments:

ChanB A vector of channels specifying boxes to be affected.

Option

'AcqStop'	Stops acquisition.
'AvgStart'	Zeroes average buffers and begins averaging.
'AvgStop'	Stops averaging.
'AvgCont'	Begins averaging, does not clear average buffers.
'AvgWait'	Waits for averaging to complete before returning averaged data.
'FEabort'	Abort Front End acquisition'
'FrmReject'	Response to Frame Accept/Reject query.
'FrmAccept'	Response to Frame Accept/Reject query.
'TestStop'	Stop a transient capture
'TrigArm'	Manually arm the trigger.

Only one of the above strings may appear on the command line.

Returns: nothing

Execute Application

'Exe'

Syntax:

```
Siglab ('Exe', filespec);
```

Purpose: to execute a program to process a file with the appropriate application.

This function usees the association established by Windows to find and run the appropriate application. The program will be started, and the call will return immediately.

Input Arguments:

filespec	Argument must contain the full parth, including extension of the file to be processed. If the file is not found, or there is no association, or the associated application is not found, then this call will do nothing.

☑ EXAMPLE

To process a document file using Word97:

Siglab ('Exe','d: \My Documents\Star Report.doc');

'Get'

Syntax:

siglab('Get','Bias');

Purpose: Determine if Bias source (ICP) is available

Input Arguments:

none

Returns:

0 if Bias Source is not installed

1 if Bias Source is installed

-1 if status not known or not ready yet.

Request Status

'GetStat'

Request overload status information

Syntax:

```
SigLab ('GetStat')  OR SigLab ('GetStatI');
```

Purpose: Requests overload status from SigLab(s). The 'GetStatI' form is useful for getting status, even if it has not changed since the last time it was requested. These calls do not return the status. The user must either poll using SigLab ('InStatus'), or rely on the status callback to return the information.

Input Parameters

'InpGain'

Syntax:

```
Scale =
siglab('InpGain',Chan,FullScale,'OptionalParameters');
```

Purpose: Sets the input channel full scale voltage range, offset, and coupling.

Input Arguments:

Chan	Channel number between 1 and 28, or a vector of channel numbers. The channels are numbered in sequence by increasing SCSI ID.
FullScale	Full Scale voltage range (must be a scalar). The available values are 10/(2^n) where n is between 0 and 9 (i.e. 10, 5, 2.5, 1.25, .625, .3125, .156, .078, .039, .0195 Volts).
'Offset',#	A number between -10 and 10. for the 5 and 10 volt input ranges, and between -2.5 and 2.5 for the 2.5 volt input range and below. Ignored if 'AC' is specified.
'AC' or 'DC' or 'Bias'	Coupling, Default is 'DC'. 'Bias' turns on the ICP source, if available.
'Cal',#	Special setting used only by calibration routine.

Returns:

Scale	The smallest available full scale setting is selected that is greater than 0.99*the requested FullScale value. Note: size(Scale) = size(Chan).

☑ EXAMPLE

1) `Scale = siglab('InpGain',1,0.4,'AC','Offset',0.555);`

Sets up channel 1 with a full scale of 0.5 volts, AC coupling, and a DC offset of 0.555 Volts. It returns Scale = 0.5 since this is the next larger attainable full scale value.

2) `Scale = siglab('InpGain',[1:4],5.2);`
Sets up channels 1 thru 4 with full scale setting of 10 volts. Scale is returned as 10.0 .

3) `Scale = siglab('InpGain',1,5,'da');`
Reports "Illegal string at 4th argument"

Acquisition Parameters

'InpSet'

Syntax:

```
Dfac = siglab('InpSet',Chan,RecSize,'OptionalParameters');
```

Purpose: Sets all data acquisition parameters

Input Arguments:

Chan	A vector of channels to be enabled.
RecSize	Input record size in samples between 1 and 8192.
'Sclock',#	Sampling clock in Hz must be 51200 for 20-22 and 20-42 systems, 12800 for 50-21 systems. If specified, **this must be** the 1st optional parameter.
'Cfreq',#	0 < Center freq. < 20000. Baseband processing used, if Cfreq is zero or not specified. If non-zero Cfreq is specified, zoom processing is used and a complex valued time history is returned.
'Srate',#	Set effective sample rate to smallest legal value > .99*#.
'BW',#	Set bandwidth to smallest legal value > .99*#.
'Dfac',#	Set decimation factor to legal value <= #.
	Note that only one of the above 3 parameters is allowed. If none of the three are supplied, max BW is assumed.

'NoFilt'	Disable decimation filters (Default = 'Filt').
'OverLap',#	Set overlap factor, # must be 0, 50, or 100 (Def=0).
'ExtClk'	Select external sampling clock.

Returns:

Dfac	Resulting decimation factor.
	The effective sample rate is Sclock/Dfac.
	The effective bandwidth is Sclock/(2.56*Dfac).

The parameters RecSize, Clock, Cfreq, Srate, BW, Dfac, Nofilt, OverLap, and ExtClk affect the box specified by the channel and not the channel itself (i.e. the channel number is used in lieu of box numbers which we have chosen not to define).

☑ EXAMPLE

1)

```
Dfac = siglab('InpSet',[1 5 6], 1000);
```

Box 1: channel 1 is enabled, channel 2 is disabled
Box 3: both channels are enabled
1000 points will be collected per frame
Dfac is returned as 1 (default)

2)

```
Dfac = siglab('InpSet',1,1000,'ExtClj');
```

Reports "Illegal string at 4th argument"

Overload Status

'InStatus'

Syntax:

```
Ovld ErrCode ErrText = siglab('InStatus');
```

Purpose: Returns instantaneous overload information.

Input Arguments: None.

Returns:

Ovld	0 if all channels are within normal range. N if channel N is overloaded. [i1 i2 ... in] if chans i1,.in are overloaded (row vector).
ErrCode (Optional)	Array of 3 error or status code values. The first element of this array, Errcode (1) will be on the following values:

0 ERR_None,	
-1 ERR_MultiBoxCable	Multibox cable not connected
-2 ERR_SyncFailure	Multiple boxes failed to sync over scsi
-3 ERR_LoadFilterCode	Input DSP fialed to read filter ucode
-4 STAT_CheckSum	Not an error: indidcates checksum to follow
-5 ERR_Discountinuous	RTW is discontinuous
-6 ERR_StackGrew	Stack Grew error! (C31 Cache bug)

-7 STAT_RPMValue	Not an error, RPM follows
-8 STAT_GainChange	AutoScaling - gain range changed
-9 STAT_Bias	Bais Info Being Passed
-10 STAT_FrameToAccept	Unused
-11 ERR_LoadOutputCode	Failed to load output microcode

ErrText (Optional) String containing error or status message (if any).

Initialization

'IOinit'

Syntax:

```
[In,Out,BW,Date,Ver]=siglab('IOinit','c:\siglab\siglab.out',D
ebugMode);
```

Purpose:

Initializes the SCSI interface connection to all the SigLabs.

Input Arguments:

The second argument specifies the file containing SigLab's code, which is downloaded if this has not already happened. See "**Error! Reference source not found.**" on page **Error! Bookmark not defined.**.

The last argument specifies the DebugMode and is optional. If it is not included, then DebugMode defaults to zero (see “Debug Mode: Debug”).

Returns:

In	The number of input channels
Out	The number of output channels
BW	Bandwidth in Hz
Date	Date string for siglab.dll
Ver	Version string for siglab.dll (e.g., '3.10')

All output arguments are optional

System Identification

'Lattice'

Syntax:

```
[p,q]=siglab('Lattice',u,y,Order,Npts,Dtype,Lambda,Epsilon);
```

Purpose: Calls the ***siglab.dll*** (host based) lattice/predictor algorithm. See *"Lattice Filters for Adaptive Processing;"* Benjamin Friedlander, Proceedings of the IEEE, Vol 70 , Number 8, August 1982.

u = Input time series.

y = Output time series.

Order = Model Order.

Npts = Number of points to use from time series.

Dtype = Data type: 0 = real, 1 = complex.

Lambda = exponential forgetting factor.

Epsilon = Initial condition (Use 1.0E-6 for best results).

Returns:

p = Numerator polynomial, length(p) = Order.

q = Denominator polynomial, length(q) = Order+1.

Burst Function Output Parameters

'OutBurst'

Syntax:

```
[On Off Cf] = siglab('OutBurst',Chan,OnTime,...
                     OffTime,'OptionalParameters');
```

Purpose: Used to set up outputs with on and off times (Arb,Rand,Chirp). Note that one of 'Rand', 'Arb', or 'Chirp' must be a parameter.

Input Arguments:

Chan	Specifies the output channel to be defined/modified.
OnTime	Desired on time in seconds. Use 0 for continous random.
OffTime	Desired off time in seconds. Use 0 for continuous random.
'Chirp',#	Specifies chirp output. # is the stop frequency.
'StartF',#	Specifies the start frequency (chirp only, default=0).
'Rand',#	Specifies random output. # is the interpolation factor. Allowed values of # are: 1,2,4,10,20,40,... 10000. These correspond to bandwidths of 20k 10k 5k 2k 1k 500 200 100 50 20 10 5 2 Hertz.
'Cfreq',#	Center frequency. Ignored for chirp. (default=0).
'Arb',#	Specifies arbitrary output. (# same as for Rand).
'Offset',#	# is starting point block offset (Arb only, default=0).

Returns:

On	Actual on time (in seconds).
Off	Actual off time (in seconds).
Cf	Actual center frequency (will be a multiple of 6.25Hz with 51200 Hz system clock)

Output Level Parameters

'OutLevel'

Syntax:

```
siglab('OutLevel',Chan,Level,'OptionalParameters');
```

Purpose: Used to set up output level, offset and ramp time parameters. If OutLevel is not called for any channel, its level will default to 1V with 0 offset.

Input Arguments:

Chan	Specifies the output channel to be set.
Level	Peak output level in volts (RMS level for random).
'Offset',#	DC Offset in Volts (default=0).
'RampTime',#	Time, in seconds (0-32), for output to ramp to its new level (default=0).

Pulse Output Parameters

'OutPulse'

Syntax:

```
FreqA = siglab('OutPulse',Chan,Freq,'OptionalParameters');
```

Purpose: Used to setup nonsinusoidal periodic outputs.

Input Arguments:

Chan	Specifies the output channel to be defined/modified.
Freq	Desired frequency.
'Square'	Select Squarewave (default). 'Triangle' 'Sawtooth', 'Impulse'

Returns:

FreqA Nearest attainable frequency.

Sine Phase Adjustment

'OutPhase'

Syntax:

```
siglab('OutPhase',Chan,Phase);
```

Purpose: Used to set relative phase between sinusoidal outputs

Input Arguments:

Chan Specifies the output channel to be defined/modified.

Phase Phase shift, in degrees.

Returns:

Nothing.

Sine Output Parameters

'OutSine'

Syntax:

```
siglab('OutSine',Chan,Freq,[Freq2 [,Ratio]]);
```

Purpose: Used to set up sinusoidal outputs (single or dual tone)

Input Arguments:

Chan Specifies the output channel to be defined/modified.

Freq Sine frequency (tone 1).

Freq2 Sine frequency (tone 2). Single tone if not included.

Ratio AmplitudeTone1 / AmplitudeTone2 (default=1).

Returns:

Nothing.

Long Record Support

'Playback'

Syntax:

```
siglab('PlayBack','OptionalParameter');
```

Purpose: Controls playback from transient capture buffer. Only one optional string parameter may be specified in a playback call.

Input Arguments:

'analog'	Return to normal analog input mode.
'memory'	Set to process data from transient capture buffer.
'nextframe'	Process next available frame (in stepmode).
'numframes'	The number of frames to be stored.
'getmem'	Prepare to fill transient capture buffer from host. This should be followed by a sequence of siglab('sendint',...) commands to do the active data transfer.
'stepmode'	Process one frame at a time.
'flowmode'	Process entire transient capture buffer once.
'circmode'	Circulate through transient capture buffer.
'wait'	Wait for position or nextframe cmd before proceeding.

'getsize'	Request the size of the transient capture buffer. Size information is actually returned by calling the siglab('CapStatus','bufsize') function.
'getstat'	Request the status of the transient capture buffer Status information is actually returned by calling the siglab('CapStatus',...) function with one of the following Operation arguments: 'discon', 'frames', 'samps', 'ovld',or 'cursamps'.
'position',#	Specify position in buffer(in samples) to start processing from.
'freebuf'	Clear transient capture buffer and free memory.
'arbcap',chans	Use transient capture buffer as source for arbitrary output. Chans argument signifies: 0 - Redirect arbitrary output to use its own buffer 1 - Use the first (or only) channel from the capture buffer for either Output 1 or Output 2. 2 - Use the second channel from the capture buffer for either Output 1 or Output 2. Not valid if there is only one channel in capture buffer. 3 - Use the two channels in the capture buffer for both Outputs. There must be two input channels, and both output channels must be set for arbitrary output with the same bandwidth. If the first argument after 'playback' is an integer in the range of 0-8, it is interpreted as a SigLab box index, with 0 being the first box. 8 signifies all boxes. If this value is not present, the command will be sent to all boxes.

Averaging Methods

'Process'

Syntax

```
siglab('Process',ChanB,'OptionalParameters');
```

Purpose: Determines the type of averaging to be performed during the measurement process.

Input Arguments:

ChanB	A vector of channels.	
'Add',n	Additive averaging (n frames), default = 'Add',1.	
	'Peak',	Peak hold averaging (n frames, forever if n=0).
	Sub',	nSubtractive averaging (n frames).
	'Exp',n	Exponential averaging (n = time constant in frames).
	'Adapt',n	Adaptive averaging (n = time constant in frames).
	'TimeAvg'	Select time domain averaging (default = 'FreqAvg').
	'OvldRej'	Reject overloads in averaged results (default = 'NoReject').
'ZeroPad'	Add Zeros to end of frame (default='NoZpad').	
'LinesAll'	Average all FFT lines in calculation for improved correlation functions (default='Lines78%').	
'Window',n	Window type n (default = 0).	

```
0=Boxcar    5=Potter210   10=BHarris61   15=Exp.1
1=Hanning   6=Potter310   11=BHarris67   16=Exp.01
2=FlatTop   7=Hamming     12=BHarris74   17=Exp.User01
3=Flat301   8=Blackman    13=Bharris92   18=Force.User
4=Flat201   9=Exact-Bl    14=Force20     19=Frq.User
```

Send Raw Command

'RawCommand'

Syntax:

```
siglab ('RawCommand', [boxnum]longarg1,
longarg2,...longargn);
```

Purpose: Send low level commands for debug purposes.

If the first argument after 'RawCommand' is an integer in the range on 0-8, it is interpreted as a SigLab box index, with 0 being the first box. 8 signifies all boxes. If this value is not present, the command will be sent to all boxes.

Arguments represent 32 bit integers to be sent to SigLab. Badly formed commands result in disaster.

One function that can be implemented using 'RawCommand' is rebooting the SigLab. Siglab('RawCommand',768) will cause a reboot.

Request Speed Data

'RpmReq'

Syntax:

```
siglab('RPMreq',mode,chan,ppr, optional argument);
```

Purpose: Sets up speed measurment modes and/or requests speed value. Despite the name, any speed values returned are in revolutions per second.

Input Arguments:

mode	0 :	stop measuring rpm
	1 :	Measure rpm from trigger pulses (time)
	2 :	Measure rpm from inst. frequency domain data on selected channel
	3 :	Measure rpm from avg. frequency domain data on selected channel
	4 :	Measure rpm from inst. DC on selected channel
	5 :	Measure rpm from avg. DC on selected channel
chan	Channel used for mode 2-5 (must be in the first box, i.e. 1 or 2 if 20-22 or 50-21, or 1-4 for 20-42)	
ppr	Number of pulses per revolution for tachometer	

Optional Arguments

'Status'	Return rpm as part of instantaneous status
'NoStatus'	[default]. Don't return instantaneous rpm. Speed will be returned with data blocks.
'OnDemand'	Return Inst. rpm only when actually requested. (Otherwise, rpm is returned to PC whenever it changes.)
'Update'	This sets OnDemand mode, and makes the request.
'Get'	Returns updated speed value, or -1 if no new value is available

Returns:

Speed in RPS for the 'Get' call, otherwise Nothing.

Arb Output Support

'SendArb'

Syntax:

```
siglab('SendArb',Data,Chan,Size,Offset)
```

Purpose: Send data block to arbitrary output data buffer in SigLab. When two channels of arb output are generated, they must have equal interpolation factors (effective sampling rates / bandwidths), equal block sizes, and equal on/off times if bursting.

Input Arguments:

Data	Data to send to the Arbitrary output data buffer. The array must contain numbers between -1 and +1. +1 corresponds to the peak Level specified in the OutLevel call.
Chan	The output channel number (1 to 14).
Size	(Optional arg) Number of elements, default = size(Data).
Offset	(Optional arg) Start sending at Data(Offset+1), default=0.

Arbcode (Optional arg)

0 = Single block of data (normal condition).
1 = Free all arbitrary output data buffers (reserved).
2 = Append to list of output buffers (for long outputs).
3 = Free all and start new list of buffers.

Calibration Support

'SendCal'

Syntax:

```
siglab('SendCal',OptionalParameters');
```

Purpose: Supports SigLab calibration (not intended for end user).

Input Arguments

'Chan',Chan	Cal factors will apply to this channel number (default=1). Must be 1st argument. Ignored for Save/Restore.
'Igain',[#]	A 10 element vector of input gain correction factors.
'Ioff',[#]	A 10 element vector of input offset correction factors.
'Idac',[#]	A 4 element vector of input offset dac cal factors.
'Output',[#]	A 18 element vector of output correction factors.
'SaveI'	Save current input cal factors in EEROM (user area).
'FactoryI'	Save current input cal factors in EEROM (factory area).
'RestoreI'	Copy input cal factors from factory area to user area.
'SaveO'	Save current output cal factors in EEROM (user area).

'FactoryO'	Save current output cal factors in EEROM (factory area).
'RestoreO'	Copy output cal factors from factory area to user area.

Send Integers to Capture Buffer

'SendInt'

Syntax:

```
siglab('SendInt',Data, Chan, Fsize, Offset, FrameNum)
```

Data	Data to send to the Transient Capture data buffer.
Chan	The output channel number (1 to 28).
Fsize	Number of elements (frame size).
Offset	Start sending at Data(Offset+1), normally = 0.
Frame	Offset in frames of Fsize elements. First frame is 0.

Set User Processing Window

'SetUserWindow'

Set user defined processing window.

Syntax:

```
siglab ('SetWindow',SlotNumber,coefficients);
```

Purpose: Used to specify a special processing window to be used with FFT's. Slots 17, 18 and 19 are reserved for user specified Exponential, Force, and standard windows, although any existing window can be overwritten.

Input Arguments:

SlotNumber — Window slot to be modified. For an exponential window, use 17. For a force window, use 18. For a standard window, use 19. Other slots may be used, at the risk of general confusion.

coefficients — Array of coefficients which define the window. This should be no more than 5 long. For a standard window, this will be the 5 point window convolution kernal. As an example, the flattop window uses: [1.0, -.970179, .653919, -.201947, .017552]. For an exponential window, the value is the exponential decay factor (e.g. .1). For the force window the value is the fraction of a frame which is passed by the window (e.g. .20)

State Space Controller Parameters

'StateSpace'

Syntax:

```
siglab('StateSpace',A, B, C, D, x0);
siglab('StateSpace','Go');
```

Purpose: Sets up State Space Controller. Siglab inputs and outputs must have been set up appropriately before making this call. Controller must be started with a subsequent siglab('StateSpace','Go') command.

Input Arguments:

A	System matrix, n by n
B	Input matrix, n by r
C	Output matrix, p by n
D	Feedthru matrix, p by r
x0	State vector, n by 1
u	Input vector, r by 1
y	Output vector, p by 1
where	
n	System order
r	Number of inputs
p	Number of outputs

Set Status CallBack

'StatusCallBack'

Syntax:

```
siglab('StatusCallBack',Func,FuncArg);
```

Purpose: Identifies Matlab function to receive asynchronous input overload status information. This need be called only once. The specified callback routine will be called whenever the overload status changes. The call will be of the form :

Func(Funcarg,Ovld), where Ovld is a vector of overloaded channels, or 0, if no overloads are detected.

Input Arguments:

Func	String containing the name of a Matlab function to receive overload status callbacks.
Funcarg	Scalar or String value to be passed to Func.

Returns:

Nothing.

Trigger Parameters

'Trigger'

Syntax:

```
Level = siglab('Trigger',ChanB,'OptionalParameters');
```

Purpose: Set up trigger conditions.

Input arguments:

ChanB	1. Dummy argument (scalar or vector). Value is ignored.	
'FreeRun'	Do not trigger, just acquire (default).	
	'Every'	Retrigger on every frame.
	'First'	Trigger on first frame only.

'AutoArm'	Automatically re-arm after a trigger (default). 'ManArm' specifies manual trigger arm (See Process Control).
'SourceI',#	Choose input channel '#' as trigger source. 'SourceO',# Choose output channel '#' as trigger source. 'SourceExt' Choose external input as trigger source.
'Delay',#	Trigger delay in samples, +/- is post/pre (default=0).
'Level',#	Trigger threshold in percent of full scale (default +25%).
'PosSlope'	Trigger slope set to positive (default). 'NegSlope' trigger slope set to negative.
'Filt'	Use digitally filtered data (default). 'UnFilt' use unfiltered data.
'LoHyst'	Use low hysteresis (default). 'HiHyst' Use high hysteresis.

Returns:

Level	Returns closest attainable trigger level to that requested.

☑ EXAMPLE

1)
```
siglab('Trigger',[1 3 5],'Every','SourceI',1,'Level',30);
```
Set to trigger on input channel 1 at a +31.25% of full scale level, positive slope, filtered data, and low hysteresis. Returns 31.25.

2)

```
siglab('Trigger',[1 3 5],'Every','Level',30);
```
Reports "Must specify trigger source."

vcap File Utilities

File Utilities

A complete set of file utilities is provided with **vcap** to aid in developing your own transient capture applications. These functions include:

vcapfile.m - read and write data to/from compact **vcap** files

vcapwthd.m - write header information to **vcap** files

vcaprdhd.m - read header information from **vcap** files

The files stored by the standard Virtual Instruments (**vos**, **vsa**, **vna**, etc.) are "normal" .mat files. As such, they can be read into MATLAB by simply using the MATLAB "load 'file.ext'" command. The important variables contained in these files are described in Section 4 of the SigLab Programming Guide located at the back of this manual.

Records captured by **vcap** are stored in more compact form. The time-domain data is stored as 16-bit signed integers rather than 64-bit double precision floating point numbers, as is standard for MATLAB. Therefore, **vcap** data files are in binary format, written and read by MATLAB using fwrite and fread commands. The format of the files is briefly described as follows:

The file header consists of a number of arrays of numbers and strings, each preceded by its dimensions. The header contains all the information needed to set up **vsa**/**vos** to process the data, including scale factors and annotation. Following the header is the actual data, which consists of frames of short integers, in channel order. For example, if 10 frames of 2 channel, 512 point data are stored, the data will be as follows:

frame 1 channel1[1 ... 512]
frame 1 channel2[1 ...512]
...
frame 10 channel1[1 ... 512]
frame 10 channel2[1 ... 512]

The file **vproc.m** is a simple example of reading in a **vcap** file which contains some speech. This speech is captured, filtered, and

sent out to the arbitrary function generator. A listing is found in Appendix C.

Routines for Reading Existing Files

vcapfile 'Open'

Syntax:

```
[fid, pathName, fileName, errmsg]=vcapfile('Open',default_path)
```

Purpose: Prompt user for file name and open an existing transient capture .

Input Arguments:

Default Path	Initial guess for file location.

Returns:

fid	error code if < 0, or the file identifier otherwise
pathname	directory specified by user for file
filename	filename specified.
errmsg	Text describing error, if any

vcapfile 'ReadHeader'

Syntax:

```
[FileStatus,NumFramesinFile,RecSizeSamps,Numchan,...
SampleRate]=vcapfile('ReadHeader',fid);
```

Purpose: Read the vcap file header and return a few essential parameters. Other file parameters can be obtained after executing ReadHeader using the 'GetAnnotation', 'GetChanInfo', 'GetVdlgs', and 'GetHdlgs' functions.

Input Arguments:

fid	File handle (File must have been opened first)

Returns:

FileStatus	If > 0 ,Offset to start of data in file, otherwise it indicates an error: -1 : file not open; -2 not a vcap file
NumFramesinFile	Number of records of size RecSizeSamps in file
RecSizeSamps	Number of samples per record
Numchan	Number of channels in file
SampleRate	Sample rate associated with data

vcapfile 'GetAnnotation'

Syntax:

```
[ FileText, TimeStamp] = vcapfile('GetAnnotation');
```

Purpose: Returns vcap file annotation. Call must have been preceeded by a 'ReadHeader' operation.

Returns:

FileText	Text string describing file (user entered)
TimeStamp	Array containing date code[optional]

vcapfile 'GetChanInfo'

Syntax:

```
[ Chanlist, Chanstat, ChanLabel, EULabel, MaxValues] =
vcapfile('GetChanInfo');
```

Purpose: Returns channel info from open vcap file. Call must have been preceeded by a 'ReadHeader' operation.

Returns:

Chanlist, Chanstat, ChanLabel and EULabel variables described in Section 4.0 (vi File Structure)

MaxValues: Vector of maximum values found for each channel[optional]

vcapfile 'GetVdlgs'

Syntax:

```
[ vdlg1_s1, vdlg2_s1, vdlg2_s1] = vcapfile('GetVdlgs');
```

Purpose: Reads vertical dialog data structures used to set up vos or vsa from vcap file. Call must have been preceeded by a 'ReadHeader' operation.

Returns:

Vertical dialog data structures

vcapfile 'GetHdlgs'

Syntax:

```
[ hdlg1_s1, hdlg2_s1] = vcapfile('GetHdlgs');
```

Purpose: Reads horizontal dialog data structures used to set up vos or vsa from vcap file. Call must have been preceeded by a 'ReadHeader' operation.

Returns:

Horizontal dialog data structures

vcapfile 'ReadFrame'

Syntax:

```
[ReadStatus , DataFrame ]=
vcapfile('ReadFrame',fid,...chanv,sampoff,size);
```

Purpose: Read a frame of arbitrary length (size) from an arbitrary position(sampoff), of one or more channels worth of data. 'ReadHeader' must be called before 'ReadFrame' to set internal variables.

Input Arguments:

fid	File handle
chanv	Vector of channels to be read ,e.g. [1 2]
sampoff	Offset in samples into file for starting value

size Number of samples to retrieve per channel

Returns:

ReadStatus -1 = file not open or valid, -2 = read beyond end of file (samples requested not available, 1 = data returned may span a discontinuity, 0 = OK

DataFrame Matrix of data from files : form is DataFrame(size, Numchans)

Routines for Writing New Files

vcapfile 'Create'

Syntax:

```
[fid, pathName, fileName, errmsg] = vcapfile('Create',...
default_path);
```

Purpose: Ask user for file name and create a new transient capture file

Input Argument:

Default Path Initial guess for file location

Returns:

fid Error code if < 0, or the file identifier otherwise

pathname Directory specified by user for file

filename Filename specified.

errmsg Text describing error, if any

vcapfile 'WriteFrame'

Syntax:

```
WriteStatus  = vcapfile('WriteFrame',fid,Fvec);
```

Purpose:Write a (multichannel) frame to a vcap file. The file must have been created, the file header written with vcapwthdr, and internal data structures set up using vcapfile('writeprepare') before this function is used.

Input Arguments:

fid File handle

Fvec arrae to write to file of form (Fvec(size,nchan))

Returns:

WriteStatus -2 if Size of fvec does does not match file record size, 0 if ok, otherwise = value returned by ferror after fwrite

vcapfile 'WritePrepare'

Syntax:

```
sfactors =
vcapfile('WritePrepare',n_chan,RecSize,vdlg1_s1,off2data)
```

Purpose: Sets up internal data structures to prepare for efficient WriteFrame operations. The file must have been created and the file header written with vcapwthdr before this function is used.

Input Arguments:

n_chan Number of channels to write

RecSize Record size, in samples

vdlg1_s1 Vertical dialog data structure

off2data Offset in file to start of data area

Returns:

sfactors Vector of scale factors for selected channels[optional]

vcapWtHdr

Syntax:

```
[OutStatus, Off2Data]= vcapWtHdr(fid, vdlg1_s1, ...

vdlg1_s2, vdlg2_s1, hdlg1_s1, hdlg2_s1, vi_timestamp,... SystemClk,
SampleRate, CenterFreq, ChanStat, ChanLabel,... EULabel,
File_Text,TCdiscon, Tcovld, NumFramesinBuf,... inTDsize, n_chan,
maxvalues );
```

Purpose:Write File Header for Transient Capture file. Most information comes from vsa/vos data structures.

Input Arguments:

fid	File Handle vdlg1_s1, vdlg1_s2, vdlg2_s1, hdlg1_s1, hdlg2_s1, vi_timestamp, SystemClk, SampleRate, CenterFreq, ChanStat, ChanLabel, EULabel Variables defined in Section 4.0 (vi File Structure)
File_Text	Text String (user supplied)
Tcdiscon	Code identifying discontinuous data
Tcovld	Code for overloads
NumFramesinBuf	Number of records to be written to file
inTDsize	Record size, in samples/chan
n_chan	Number of channels in file
MaxValues	Vector containing max value in volts for each channel

vcapRdHdr

Syntax:

```
[Off2data, vdlg1_s1, vdlg1_s2, vdlg2_s1, hdlg1_s1, hdlg2_s1,...

vi_timestamp, SystemClk, SampleRate, CenterFreq,...

ChanStat, ChanLabel, EULabel, File_Text,TCdiscon, TCovld,...

NumFramesinBuf,inTDsize, n_chan, Maxvalues ] = vcapRdHdr(fid);
```

Purpose: Read File Header for Transient Capture file. Most information is in form ofvsa/vos data structures. Most of this is not needed, except to set up vcap.mi. Normally the user will use vcapfile('ReadHeader',...) instead of this. File must have been

opened (using vcapfile('Open',...) or fopen(...) before calling this routine.

Input Argument:

fid File Handle

Returns:

Off2data	Offset in file to start of data if > 0; if -1 : file not open, if -2: not a vcap file vdlg1_s1, vdlg1_s2, vdlg2_s1, hdlg1_s1, hdlg2_s1,... vi_timestamp, SystemClk, SampleRate, CenterFreq,... ChanStat, ChanLabel, EULabel Variables defined in Section 4.0 (vi File Structure)
File_Text	Text String (user supplied)
Tcdiscon	Code identifying discontinuous data
Tcovld	Code for overloads
NumFramesinBuf	Number of records to be written to file
inTDsize	Record size, in samples/chan
n_chan	Number of channels in file
MaxValues	Vector containing max value in volts for each channel

Section 4.0
FILE STRUCTURE REFERENCE

SigLab Measurement Data Structure Reference

Each VI is capable of control state and measurement storage. Typing ***help vxx*** at the MATLAB prompt (e.g. ***help vna***) will display the variables stored to ***vna*** files and their definitions. The following reference material is provided as an additional guide for reading and interpreting the variables stored in the VI-generated files. The variables of interest to the user are always named with a combination of upper and lower case letters.

MATLAB 5.x enables data structures. The v3.0 ***vos, vsa***, and ***vna*** SigLab measurement applications use a new SigLab Measurement data structure – SLm.

The SLm structure reference for vos, vsa and vna

The SLm structure is used by the v3.0 versions of ***vos***, ***vsa***, and ***vna*** applications as a container for measurement data and associated setup information. It is accessed in two ways:

1. Via loading a measurement file that has been saved with v3.0 code,
   e.g.`load file_name.vna -mat`

2. Via an a 'get','meas' call to the active application
   e.g. `SLm = vna('get','meas'); % vos, vna, vsa return SLm`

The members of this data structure are described briefly below. A more detailed discussion of the structure and examples of its uses can be found in Section 2.0 of this manual. In the descriptions below, parameters are scalar unless identifed as vectors or structures. The following abbreviations will be used for conciseness:

ch	Channel number, in the range of 1..16. Used to specify to which channel a parameter or data vector applies
refch	Reference channel number, in the range of 1..4. Specifies the reference channel for a cross channel measurement
respch	Response channel number, in the range of 1..16. Specifies the response channel for a cross channel measurement
fdsize	Frequency domain frame size. The number of points in a frequency domain function such as aspec or xfer. This value is normally equal to (tdsize/2.56)+1.
tdsize	Time domain frame size. The input record length, in samples.
SLm	An instance of the Slm data structure

SLm =

scmeas - ***Vector of structures containing single channel measurement related info***

tdmeas	Input channel time domain data (vector, time domain, volts), a vector containing *tdsize* samples, or empty if no time data is stored for this channel Usage: time_block = Slm.scmeas(*ch*).tdmeas;
aspec	Autospectrum (vector, frequency domain, vrms^2), a vector containing *fdsize* values, or empty if no data is stored Usage: data = Slm.scmeas(*ch*).aspec
fft	FFT (vector, frequency domain, volts), a vector containing fdsize values, or empty if no data is stored. usage: data = scmeas(*ch*).fft;
acor	Auto correlation (vector, time domain, volts^2) a vector containing *tdsize* values, or empty if no data is stored. Usage: data = scmeas(*ch*).acor;
label	Channel label string Usage: Labeltext = scmeas(*ch*).label;
eu_on_off	Engineering unit state, 0= off, 1= on
euscale_fac	For infernal use only
eu_string	Engineering unit label string, e.g. "g's"
eu_val	Engineering unit scale factor, in eu/volt Example: if Slm.scmeas(*ch*).eu_on_off, scaledTimeData = SLm.scmeas(*ch*)*SLm.scmeas(*ch*).eu_val; disp(['scaledTimeData is in units of ',SLm.scmeas(*ch*).eu_string]); end;
fs_val	Full scale voltage range, e.g. 10.0
db_ref	0 db reference value, in volts (typically 1.0)

xcmeas	vector of structures containing cross channel measurements. One instance per possible channel pair (reference channels 1 to 4, response channels 1 to 16). Only the instances for channel pairs (reference versus response) for which data is stored will be filled, others will be empty..

Fields within xcmeas structure

xfer	Transfer function , (vector, frequency domain, volts/volts), a vector containing *fdsize* complex values, or empty if no data is stored Usage: data = Slm.xcmeas(*refchan*, *respchan*).xfer
coh	Coherence function, (vector, frequency domain, unitless), a vector containing *fdsize* values, or empty if no data is stored Usage: data = Slm.xcmeas(*refchan*, *respchan*).coh;
cspec	Cross Spectrum, (vector, frequency domain, volts*volts), a vector containing *fdsize* complex values, or empty if no data is stored Usage: data = Slm.xcmeas(*refchan*, *respchan*).cspec;
ccor	Cross Correlation, (vector, time domain, volts), a vector containing *tdsize* values, or empty if no data is stored Usage: data = Slm.xcmeas(*refchan*, *respchan*).ccor;
imp	Impulse Response function, (vector, time domain, v), a vector containing *tdsize* complex values, or empty if no data is stored Usage: data = Slm.xcmeas(*refchan*, *respchan*).imp;

xcstate Cross channel state structure

Fields within xcstate structure

refc	vector of reference channels (defines legit values for ref below)
resp.r	vector of structures listing response channels for each possible reference channel (i.e., 1..4) for example SLm.xcstate.resp(1).r provides list of response channels associated with refernce channel #1 ref = ref channel (1..4) i = response channel (1..16) from SLm.xcstate.resp(ref).r
clist	List of channels enabled
tdxvec	Time vector. Represents the sampling times in seconds, normally uniformly spaced The length of the vector, *tdsize*, is the record size. Used in plotting time functions. For example, to plot the time history of channel 1, one would use: plot(SLm.tdxvec,SLm.scmeas(1).tdmeas);
fdxvec	Frequency vector. Represents the frequency values in Hertz for the frequency domain measurements. It is of length *fdsize*, which is the number of lines in the spectrum
clist	Channel list vector. Variable length vector which identifies all channels which were enabled . For example, if channels 1, 2 and 3 were enabled then Slm.clist would contain [1 2 3].:
numin	Number of input channels in system
wincor	Correction factor for analysis window
winsel	window (code) used for analysis. see siglab('process') for code values);
rbw	Resolution in hertz
navg	Number of averages completed

zpad	Zero padding flag (1 if padding on)
ovld	Overloaded channel list. If no channels were overloaded during the measurement, it will be a scalar equal to zero. If one or more channels had an input overload condition, the channel number will appear in the list.
zoomcf	Zoom center frequency in hz (0 if baseband)
filestor	file storage information struct (obscure)

Fields within filestor structure

label	10 element cell array with labels for vna file storage dialog
state	10 element cell array which identifies which functions may be present in file
fields	9 element cell array with names for functions ('tdmeas','aspec',...)

File Structure for vss, vid, vda and vcap

Each VI is capable of control state and measurement storage. Typing ***help vxx*** at the MATLAB prompt (e.g. ***help vda***) will display the variables stored in these files and their definitions. The following reference material is provided as an additional guide for reading and interpreting the variables stored in the VI generated files.

Engineering Units

The measurement data is always stored without the engineering units applied to the data. The user can apply (or not) the engineering units stored in the ChanStat variable to the data.

Window Correction Factor

Auto spectrum measurements are always stored as Volt squared rms. To obtain the proper results for a power calculation, the spectral data must be multiplied by the window (as in Hanning) correction factor.

System Related

SystemClk

Represents the fundamental system sampling clock. The default value for this is 51200 Hz.

Size: real, scalar.

☑ EXAMPLE

```
SystemClk = 51200;
```

vi_timestamp

A six element vector representing the time that the measurement was acquired. The format is:

```
[year month day hour minute second]
```

☑ EXAMPLE

```
vi-timestamp=[1994 12 31 23 59 59.9] (New Years Eve)
```

Acquisition

SampleRate

Represents the actual sampling rate (after decimation) of the input subsystem data stream.

Size: real, scalar.

☑ EXAMPLE

```
SampleRate = SystemClk/20; % system sampling clock after %
decimate by 20 filter
```

CenterFreq

Represents the center frequency selected when the measurement was made. This parameter relates to measurements made with zoom bandpass filtering. If the CenterFreq is 0.0, the data is from a base band measurement. If it is non-zero, a bandpass zoom measurement was made.

Size: real, scalar

☑ EXAMPLE

```
CenterFreq = 0.0; % a base band measurement
```

ChanStat

Defines the status of the channels involved with measurements. It is used with both single and dual channel measurements.

Channel:	The channel number involved with the measurment.
EU on/off:	Engineering Units On(1) or Off(0).
EU Value:	The value entered by the user for the engineering unit.
OvldStatus:	Channel overload flag 1=no overload, -1=overload.
VFS:	Full scale measurement range in Volts.

Size: real, 1..NCH x 5 (NCH = # of channels with measurements stored).

☑ EXAMPLE

```
%           Channel  EU on/off  EU value   OvldStatus   Vfs
  ChanStat   = [1       0         1.0          1          10;
                2       0         1.0          1           5;
                5       1         0.1234       1         2.5;
               11       1         5.4321      -1         2.5];
```

ChanLabel

Contains channel labels. The end of the field is delimited by the ~ (tilde) character.

Size: NCH x 20.

☑ EXAMPLE

```
ChanLabel =   ['Channel 1~          ';
               'Power Ch2~          ';
               'Torque Ch5~         ';
               'Current Ch11~       '];
```

EULabel

Contains Engineering Unit labels.

Size: NCH x 20.

☑ EXAMPLE

```
EULabel   =  ['Gs~                  ';
              'Watts~               ';
              'Inch Pounds~         ';
              'Amps~                '];
```

Navg

Represents the number of averages used to produce a measurement. If Navg is 0 the data in TimeDat or AspecDat is unaveraged. If it is >0, the data has been acquired with averaging.

Size: real, scalar.

☑ EXAMPLE

```
Navg      = 100;
```

Output

SampRateOut

Represents the actual sampling rate (before interpolation) of the output data stream. The effective bandwidth of the output data is this number divided by 2.56.

☑ EXAMPLE

```
SampRateOut = SystemClk/10; % sampling clock before
                            % interpolate by 10 filter
```

CenterFreqOut

Represents the output modulator center frequency. If it is 0.0, the output is not translated. The translated output is double-sided unlike the input zoom processing which produces a single-sided spectrum with a complex data stream.

Size: real scalar.

☑ EXAMPLE

```
CenterFreqOut = 2500.0; % the spectrum of the output will be
                        % centered around this frequency.
```

Time-Domain Data

TrigDelay

Defines the trigger delay in samples. If it is < 0, the acquisition used pre-trigger, if > 0 it was post-triggered.

Size: real, scalar.

Units: Samples.

☑ EXAMPLE

```
TrigDelay  =  -20; % data starts 20 samples
                   % before the trigger event (pre-trigger)
```

Tvec

Vector defining the sampling time instants in seconds. This vector is always provided with time domain measurements.

TDL = Time Domain record length.

Size: real, TDL x 1.

Units: Seconds.

☑ EXAMPLE

```
Tvec      = (TrigDelay:1:79)'*(1/SampleRate);
                        % the acquisition had 20 samples
                        % of pre-trigger, with a buffer
                        % length of 100 samples (79-(-20)+1).
                        % TDL = 100;
```

TimeMap

Defines channel / data relationship for TimeDat .

NTDI = Number of active Time Domain Input channels.

Size: real, 1..NTDI x 1.

☑ EXAMPLE

```
TimeMap  =  [1;2;5];
```

TimeDat

Contains time domain measurements. It is also used to store waveforms for arb-out. The array is complex if the data is acquired with zoom enabled (CenterFreq ~=0).

Size: real or complex TDL x NTDI.

Units: Volts.

☑ EXAMPLE

```
% mock data for channels given the above TimeMap
```

```
TimeIDat  =  [sin(50*2*pi*Tvec),... % channel 1
              cos(25*2*pi*Tvec),... % channel 2
              100 * Tvec.*Tvec];    % channel 5
```

Frequency Domain Data

Fvec

Defines the frequency values in Hz for the measurements in the vectors AspecDat, XferDat, CohDat, and CspecDat. This vector is always provided with frequency domain measurements. The frequency separation of the points can be unequal, but the data will always be monotonically increasing.

$$Fvec(n+1) > Fvec(n)$$

Size: real, FDL x 1; FDL = Frequency Domain buffer Length.

Units: Hertz.

☑ EXAMPLE

```
if CenterFreq == 0.0,
    Fvec = (0:1:1024/2.56)*(SampleRate/1024);
else
    Fvec = CenterFreq +...
           (-512/2.56:1:512/2.56)*(SampleRate/1024);
end;
```

NonUniform

Indicates if Fvec has a uniform element spacing (Future).

If NonUniform=1, non-uniform spacing.

Size: real, scalar

☑ EXAMPLE

```
UniformFlg  = 1;
```

WindowCor

The first element of WindowCor defines the power/amplitude correction factor for the window used (if any) in the spectral estimation. The second element is the number between 0 and 16

specifying the window type. See the SigLab process command for the definition of the window type. The Aspec is always scaled to have proper amplitude values. If you want the Aspec to be scaled properly for power density, you must multiply it by this correlation factor.

Size1: real, NPTx1.

Units: None.

The second size option would be NPTxNCH which allows a different window to be used on each channel (very rare).

Size2: real, NPT x NCH.

WinName

Contains window names. The end of the field is delimited by the ~ (tilde) character (Future).

Size: NCH x 20.

☑ EXAMPLE

```
WinName   =  ['BoxCar~             ';
              'Hanning~            ';
              'FlatTop~            ';
              'Hamming~            '];
```

AspecMap

Defines channel / data relationship for AspecDat.

Size: real, NAS x 1 (NAS = # of Aspec measurements).

☑ EXAMPLE

```
AspecAMap =   [5;11];
```

AspecDat

Contains auto-spectrum measurements at the frequencies defined by Fvec. This is a single channel measurement and the data in AspecMap determines the channel / data pairs.

Size: real, FDL x NAS.

Units: Volts squared rms.

XferMap

Defines the channels used to measure the transfer functions in XferDat. Each of these measurements requires a reference channel and a response channel.

Size: real, 1..NXF x 2 (NXF = # of transfer function measurements).

☑ EXAMPLE:

```
%                resp.   ref
  XferMap =      [2,      1;
                  5,      1;
                  11,     5];
```

XferDat

Contains transfer function measurements at the frequencies defined by Fvec. This is a two channel measurement and the data in XferMap determines which channel was the reference and which was the reponse.

Size: complex, FDL x NXF.

Units: Volts/Volt.

CohMap

Defines the channels used to measure the Coherence functions in CohDat. Each of these measurements requires a reference channel and a response channel.

Size: real, NCHF x 2 (NCHF = # of coherence function measurements).

☑ EXAMPLE

```
%                resp.   ref
  CohMap  =      [2,      1;
                  5,      1;
                  11,     5];
```

CohDat

Contains coherence function measurements at the frequencies defined by Fvec. This is a two channel measurement and the data in CohMap determines which channel was the reference and which was the reponse.

Size: real FDL x NCHF.

**CspecDat*

Contains cross-spectrum function measurements at the frequencies defined by Fvec. This is a two channel measurement and the data in CspecMap determines which channel was the reference and which was the reponse.
*CspecDat is not saved in your setup file. However, it can be formed as follows:

```
CspeDat = XferDat .* AspecDat (:,1);
```

🕮 NOTE

All transfer function measurements are stored with the autospectrum of the reference channel. To obtain the cross spectrum, multiply the transfer function by the autospectrum.

Size: complex, FDL x NCSF.

Vid_Poles

Contains vector of pole locations in s- or z-plane.

Vid_Zeros

Contains vector of zero locations in s- or z-plane.

VCAP File Structure

The long records captured by the vcap Transient Capture Program are stored in a more compact form than that used for VI files. The time-domain data are stored as 16-bit signed integers rather than 64-bit double precision floating point numbers, as is standard for MATLAB. Therefore, vcap data files are in binary format, written and read by MATLAB using fwrite and fread commands. The format of the files is briefly described as follows.

The file header consists of a number of arrays of numbers and strings, each preceded by its dimensions (m and n) . The header contains all the information needed to set up vsa/vos to process the data, including scale factors and annotation. The first 2 sections of the header contain version-related items and vsa/vos related items, and are not actually needed to interpret the data. They are used by vcap to set up the proper state for vsa or vos. The third section contains various essential parameters needed to describe the data stored in the file. Note that even if header items are not used, it is necessary to read them (or at least their dimensions), in order to locate the items of interest in the header, and to locate the data.

The vcap file utilities **vcapfile**, **vcapwthdr**, and **vcaprdhdr** handle the details of accessing items in the file header, and extracting data, so it is normally not necessary to understand the minutia of vcap file organization. However, for those gluttons for punishment, the following tables describe the file structure in more detail.

Version Related Items

Name	Type	Description
n	short	Length of key string
key	string	File type string(‘DSPT vcap ‘)
n	short	Length of capver string
capver	string	Version string for vcap s/w (e.g. ‘1.15’)
n	short	Length of filever string
filever	string	Version string for vcap file. Normally the same as capver.
n	short	Length of codedate string
codedate	string	Date string for SigLab code used to capture data (e.g. ‘07\22\95 08:09’)
n	short	Length of codever string
codever	string	Version string for SigLab code (e.g. ‘1.15’)
n	short	Length of matver string
matver	string	Version string for MATLAB (e.g. ‘4.2c.1’)
n	short	Length of sparetext string
sparetext	string	Unused string
spareval	double	Unused

vos/vsa Dialog Support Items

Name	Type	Description
m	short	Row dimension for vdlg1_s1 vector
n	short	Column dimension for vdlg1_s1 vector
vdlg1_s1	double vector	vos/vsa vertical dialog numerical states
m	short	Row dimension for vdlg1_s2 vector
n	short	Column dimension for vdlg1_s2 vector
vdlg1_s2	string vector	vos/vsa vertical dialog string states
m	short	Row dimension for vdlg2_s1 vector
n	short	Column dimension for vdlg2_s1 vector
vdlg2_s1	double vector	more vos/vsa vertical dialog numerical states
m	short	Row dimension for hdlg1_s1 vector
n	short	Column dimension for hdlg1_s1 vector
hdlg1_s1	double vector	vos/vsa horizontal dialog numerical states
m	short	Row dimension for hdlg1_s2 vector
n	short	Column dimension for hdlg1_s2 vector
hdlg1_s2	double vector	vos/vsa horizontal dialog string states
m	short	Row dimension for sparevector (1)
n	short	Column dimension for sparevector (4)
sparevector	double vector	Unused

vcap Specific Items

Items marked with an * are also used in VI files, and are described in more detail under VI File Structure.

Name	Type	Description
m	short	Row dimension for vi_timestamp vector
n	short	Column dimension for vi_timestamp vector
vi_timestamp*	double vector	Time at which file was written - [year month day hour minute second]
SystemClk*	double	System sampling clock - normally 51200.
SampleRate*	double	Actual (effective) sample rate, after decimation.
CenterFreq*	double	Selected center frequency, if zoom; otherwise = 0.
m	short	Row dimension for ChanStat vector
n	short	Column dimension for ChanStat vector
ChanStat*	double vector	Defines status of channels. Includes channel number, gain, EU and overload status. See VI file structure for details
m	short	Row dimension for ChanLabel vector
n	short	Column dimension for ChanLabel vector
ChanLabel*	string vector	Contains Channel labels. See VI file structure for details
m	short	Row dimension for EULabel vector
n	short	Column dimension for EULabel vector
EULabel*	string	Contains Engineering unit labels. See

	vector	VI file structure for details
n	short	Length of File_Text string
FileText	string	Contains user supplied annotation (i.e. Notes)
TCdiscon	double	Continuity status. Set to 1.0 if data contains gaps.
TCovld	double	Overload status. When converted to integer, bits indicate overloads on input channels. Bit 0 for chan 1, bit 1 for chan 2 etc.
NumFrames	double	Number of frames (of size inTDsize) stored in file
m	short	Row dimension for maxValues vector
n	short	Column dimension for maxValues vector
maxValues	double vector	Contains maximum value, in volts, for each channel in file if values are non-zero.
inTDsize	double	Record size, in samples
nChan	double	Number of channels stored

The vcapWtHdr function will return the offset in the file to just past the nChan item, which is where the data will begin (Off2Data).,

vcap File Data area

The time-domain data in the Data area is stored in short integer format, in arrays of size inTDsize, in channel order. A typical file, with 2 channels of data (nChan = 2), and 10 frames of data (NumFrames = 10), using a record size of 1024 (inTDsize = 1024), Would be organized as follows:

Frame 1, channel 1[1..512] offset in data area = 0

Frame 1, channel 2[1..512] offset in data area = 1024

Frame 2, channel 1[1..512] offset in data area = 2048

Frame 2, channel 2[1..512] offset in data area = 3072

...

Frame 10 channel 1[1..512] offset in data area = 18432

Frame 10 channel 2[1..512] offset in data area = 19456

The position in the file (in bytes, as used with fseek) for Frame number F, channel number C =

Off2Data + ((F-1) * (inTDsize * 2 * nChan)) + ((C -1) * inTDsize).

This applies to baseband data; in the case of Zoom data, multiply inTDsize by 2 wherever used.

Section 5.0
MPP: A MATLAB PREPROCESSOR

Synopsis

Pre-processors are part of most modern languages for good reasons, however MATLAB does not come with one. **MPP** was designed to fill this void. An example is given to show how to make a MATLAB GUI application more readable and easier to modify while significantly speeding up the initial compile and load time. The execution speed is also often improved.

An important feature of MPP is that line numbers are preserved. Thus when MATLAB reports an error at line #n, the error can be corrected by going to line n in the preprocessor source file.

MPP is currently available for the PC, DEC stations (Ultrix), and for Sun workstations.

Motivation

With the advent of the GUI tools in MATLAB v4, programs are quickly growing in size and complexity. Hundreds of constants may be needed to represent screen positions, handles indexes and other program elements. If all these constants are sprinkled throughout the code, it quickly becomes unreadable and difficult to modify. The solution to this problem is to define variable names for each constant required at the beginning of the code (or in a separate M-file script). For example:

```
slider_width = .2; % make all sliders the same width
```

Besides making the program more intelligible, this definition makes it trivial to change the slider width if that is ever required. This is nothing new. It's the commonly accepted style in any programming language. In fact, most languages provide a preprocessor to make it easy and efficient to define literals to represent constants (e.g. the #define statement in C).

Since MATLAB doesn't have a preprocessor, a variable must be defined for each constant. However once there are many of these constants, a penalty is paid in compile and execution time. MATLAB's interpretive environment makes this penalty worse. This encourages the less readable style of including constants directly in the M-file. The solution is to use a preprocessor with MATLAB.

MPP is the result of DSP Technology’s effort to provide such a solution for MATLAB. Two major features of MPP which precluded using a preprocessor designed for other languages are:

1. The preprocessor constructs are designed in such a way that the source code can be run by MATLAB whether or not the source was passed through MPP.
2. The number of lines in MPP's output file is the same as the number of lines in its input file. Thus when a MATLAB error message reports a line number, it is easy to go to the proper place in MPP's input.

In a test case, a large MATLAB GUI program was able to compile and load twice as fast after using MPP. Execution speed increased significantly as well.

MPP (in the PC environment) is a DOS application. To use MPP one of three options must be used:

1. mpp.exe located in the current directory.
2. mpp.exe on the DOS path.
3. An explicit path be supplied to invoke MPP.

Calling Sequence

The calling sequence for MPP is:

```
MPP FileIn [FileOut]
```

*The input file to MPP normally uses the extension '**.mi**'*

where:

```
FileIn = Full path and file name
```

for MPP input (required argument). If the extension is omitted, **.mi** is assumed. If no path is given, the local directory is assumed.

The optional output file argument is defined by:

```
FileOut = Full path and file name
```

*The output file from MPP normally uses the '**.m**' extension.*

If the extension is omitted, **.m** will be assumed.

If the output argument is not included, then MPP will write its output to a file with the same name as FileIn except that the extension is changed to **.m**. If the input file already has **.m** as its extension, then MPP will output its results to **mpp.m**.

MPP makes a single pass scan through input file FileIn and passes all characters to output file FileOut unmodified except in the following four situations:

1. Include Files.
2. Literal assignments.
3. Literal substitutions.

Include

When the string

```
%include
```

is found, an include section is begun. This string may be anywhere on a line, but it must be exactly as shown above. Thus

```
% include or
%Include
```

does not initiate an include section. The form of an include section is as follows:

```
%include
  include1; % comments are allowed here
  include2; % refers to include file named "include2.m"
  includeN;
%end_include
```

To find the include file, MPP will first look for the specified file (with a .m extension) in the current directory. If that file is not found, MPP will look for a list of include directories in a file called MPP.INI in the local directory. If MPP.INI is not in the local directory, MPP will try to find it in the same directory that contains MPP.EXE. Once MPP.INI is found, the first line of that file is read, which should contain a string such as:

```
c:\matlab\siglab\vcom
```

MPP will search the specified directory for the include file. If it doesn't find it, the second line of MPP.INI will be read. If the end of MPP.INI is reached before finding the include file, an error message is produced.

Include files may contain only three types of lines:

1. Comment lines.
2. Blank lines.
3. Lines of the form token = ReplacementText.

The last form will be treated in the same manner as lines within a define section as described in the following literal assignments description. The replacement text may span multiple lines by using the MATLAB line continuation symbol (three consecutive dots).

The

```
%include
```

and

```
%end_include
```

lines are passed to the output file, but the include file names in between are not. If the first line of the include file is a comment (which is a good practice), then that line will be inserted into the output file in place of the include file name (this preserves line numbers). Otherwise the include file name is replaced with a line of the form:

```
% IncludeFilename.m;
```

Literal Assignments

A define section is begun when an include file is opened for input or when the string "%define" is found. An example of a define section follows:

```
%define
  literal1 = ReplacementText1; %comments are allowed here
  literal2 = ReplacementText2;
  literal3 = ReplacementText3;
  literal4 = literal1(1,a,literal2);
  literal5 = 10 * literal2 + literal3;
%end_define
```

A literal can be any contiguous string of alphanumeric characters including the underscore (_) that does not begin with a digit. In addition, the $ character is allowed if it's the first or the last character of the literal (the reason for this is explained in the section covering *Pre-defined literals and their use*). Also the left parenthesis character is allowed as the last character of a literal to allow function definitions (see the last example in examp0.mi). The replacement text for the literal consists of all the characters after the equal sign until the end of the line (excluding the white space after the = and at the end of line). Comments are excluded as well. The replacement text is optionally terminated by a semicolon which is not considered part of the replacement text. The replacement text may include any character except "%" and ";", although ";" is allowed if it's used as a separator between elements of an array. If these characters are needed as replacement text in other circumstances the predefined literals can be used. If the replacement text contains a literal that is previously defined within a define section, then that literal is replaced with its previously defined replacement text. For example, the replacement text assigned to literal4 above would be:

```
ReplacementText1(1,a,ReplacementText2)
```

In a limited set of situations MPP will also do arithmetic simplifications. For example, if ReplacementText2 and ReplacementText3 were both numbers, then the replacement text assigned to literal5 will be reduced to a single number. MPP will simplify the replacement text if the expression after the equal sign consists only of these 5 operators:

1. Addition +
2. Subtraction -
3. Multiplication *
4. Division /
5. Association ( )

Also, all the operands must be numbers, literals which have previously been assigned to numbers, or expressions which have been reduced to numbers. If any of these conditions are not met, then MPP will simply associate the literal with the unmodified replacement text.

If the define section was part of the original MPP input file (i.e. it was not in an include file), then while MPP is saving the literal assignments it also passes the define section through to the output file modified by inserting an extra "%" character at the beginning of the line. Thus in the example above the following text would be output.

```
%%define
  %  literal1 = ReplacementText1;%comments allowed here
  %  literal2 = ReplacementText2;
  %  literal3 = ReplacementText3;
  %  literal4 = literal1(1,a,literal2);
  %  literal5 = 10 * literal2 + literal3;
%%end_define
```

When MPP opens an include file, it treats the contents as a define section in a similar way to that described above except that no text is passed on to the output file while the literal assignments are being made. This behavior assures that the number of lines in InFile will be the same as the number of lines created in OutFile. This is necessary because of MATLAB's use of line numbers for locating errors.

Literal Substitutions

When MPP encounters a “literal” in the input stream that has been previously defined in a define section, then MPP outputs the replacement text to OutFile instead of the original literal. There are two exceptions to this rule: 1) MPP does not replace the literal with its replacement text if the literal is within a comment. (i.e. is on a line following a %), and 2) MPP does not replace literals that are within a quoted string. A quoted string is a group of characters which begins with a ' and ends with an odd number of consecutive ' characters. For example, the following is a quoted string:

```
'set(gcf,''WindowButtonUpFcn'','''');'
```

Assignment Operator Simplifications
Even outside of a define section, when the MPP encounters text of the form:

```
literal = expression;
```

where expression meets all the conditions specified above for the define sections, then MPP will replace the expression with a single number. For example, if InFile contains the string:

```
if val == 0  xy = 2/5; else xy = 1+5/2;
```

Then the following text will be written to OutFile:

```
if val == 0  xy = 0.4; else xy = 3.5;
```

Note that this is the only case where expressions are simplified outside of define sections. Thus the line:

```
if val == 0  xy = sqrt(1+5/2);
```

would not be simplified (Possible future extension to MPP).

Handling the Vector Operator "[ ]"
Since the vector operator is not listed among the five expression operators, you would expect a left or right bracket to prevent any expression containing it from being evaluated. While this is true in general, there are a few special cases that are so useful, that MPP was made to recognize this operator in those situations. The first of these special cases allows MPP to simplify expressions which are elements of a vector within a define section. For example:

```
%define
  vec1 = [3, 2*2, 5+1];
%end_define
```

In this case, the replacement text for literal vec1 would be:

```
[3, 4, 6]
```

The delimiters allowed between the elements of vectors defined within a define section are the comma (for row vectors) and the semicolon (for column vectors). Unlike MATLAB, you may not use spaces or newline characters to delineate the elements of a vector within a define section.

Note that MPP will not simplify vectors occurring outside of a define section.

The second of these special cases allows MPP to extract an element of a previously defined vector. Thus if vec1 is defined as above, then vec1(2) will be replaced with the number 4. This replacement would take place wherever vec1(2) occurs, both inside and outside of define sections.

See the following example to understand how MPP treats vectors.

Although literals can be defined which contain nested brackets, MPP will not simplify expressions within such definitions. This also prevents MPP from extracting a single element from the vector. If this is done, unintended results may result. For example, if a line of a define section is:

```
a = [3, 4, [5, 6]];
```

If a(3) is referred to later in the code, MPP will substitute "[5, 6]." This is not what was intended, since in MATLAB [3, 4, [5, 6]] is a different way of writing the 1 by 4 vector: [3, 4, 5, 6];.

An Example ☑

Suppose the file examp0.mi contains the following:

```
%define
  pi = 3.1415926;
  e = 2.7182818284590;
  big = 1e20 * pi * e;
  a = 3;
  car = 9999;
  i = -((2.0 - 1e4 + car) * a + 1) * 10 * 10 / 2;
  b = [2*3+a, 4*5, 6*7+10, car];
  x = [10.1, 10.2, 0.00001*a, 1e4]; % note that the
                                    % leading 0 is needed
  v = [b(2), x(3), b(2)*x(3)];
  labels = ['Label01'; 'Label02'; 'Label03'; 'Label04'];
  bx = b + x;% this will work,
              % but next line will be far more efficient
  bxx = [b(1)+x(1), b(2)+x(2), b(3)+x(3), b(4)+x(4)];
        % see the expansion of this
  PHASE( =(180/pi) * angle(; %phase in degrees
%end_define

%  yy = big;
   yy = big;

%  ii = i;
   ii = i;

%  bbb = b;
   bbb = b;

%  bbb = b(4) * b(1);
   bbb = b(4) * b(1);

%  xx = x(1) + aaa + x(3) + x(2);
   xx = x(1) + aaa + x(3) + x(2);

%  xx = x(1) + x(3) + x(2);
   xx = x(1) + x(3) + x(2);

%  x4 = v * v(3);
   x4 = v * v(3);

%  ebx = bx;
   ebx = bx;

%  ebxx = bxx;
   ebxx = bxx;

%  ee = 2e3 + 2.0e-2 * 3.e2;
   ee = 2e3 + 2.0e-2 * 3.e2;

%  xlabel([labels(2),labels(4)]);
   xlabel([labels(2),labels(4)]);

%  if q1 == q2  ex1 = 3/4; else ex1 = 4/3; end;
   if q1 == q2  ex1 = 3/4; else ex1 = 4/3; end;
%  deg = PHASE(xx + yy*j)
   deg = PHASE(xx + yy*j)
```

After typing "MPP examp0" at the DOS prompt, the file examp0.m will be written as follows:

```
%define
  %pi = 3.1415926;
  %e = 2.7182818284590;
  %big = 1e20 * pi * e;
  %a = 3;
  %car = 9999;
  %i = -((2.0 - 1e4 + car) * a + 1) * 10 * 10 / 2;
  %b = [2*3+a, 4*5, 6*7+10, car];
  %x = [10.1, 10.2, 0.00001*a, 1e4]; % note that the
                                     % leading 0 is needed
  %v = [b(2), x(3), b(2)*x(3)];
  %labels = ['Label01'; 'Label02'; 'Label03'; 'Label04'];
  %bx = b + x;% this will work,
               % but next line will be far more efficient
  %bxx = [b(1)+x(1), b(2)+x(2), b(3)+x(3), b(4)+x(4)];
   %see expansion of this
%end_define

%  yy = big;
   yy = 8.539734077001e+20;

%  ii = i;
   ii = -200;

%  bbb = b;
   bbb = [9,20,52,9999];

%  bbb = b(4) * b(1);
   bbb = 89991;

%  xx = x(1) + aaa + x(3) + x(2);
   xx = 10.1 + aaa + 3e-05 + 10.2;

%  xx = x(1) + x(3) + x(2);
   xx = 20.30003;

%  x4 = v * v(3);
   x4 = [20,3e-05,0.0006] * 0.0006;

%  ebx = bx;
   ebx = [9,20,52,9999] + [10.1,10.2,3e-05,10000];

%  ebxx = bxx;
   ebxx = [19.1,30.2,52.00003,19999];

%  ee = 2e3 + 2.0e-2 * 3.e2;
   ee = 2006;

%  xlabel([labels(2),labels(4)]);
   xlabel([ 'Label02', 'Label04']);

%  if q1 == q2  ex1 = 3/4; else ex1 = 4/3; end;
   if q1 == q2  ex1 = 0.75; else ex1 = 1.333333333333; end;
%  deg = PHASE(xx + yy*j)
   deg = (180/3.1415926) * angle(xx + yy*j);
```

Pre-defined Literals and Their Use
The following three literals are pre-defined since syntax prevents defining them within a define section:

Literal	Value	Use
c$	%	comment character
s$	;	statement terminator
$$	'	quote character
FileName$		MPP's input file (without extension)

1. The comment character can be used to selectively comment out debugging code, or even as a way of conditional compilation. For example:

```
%define
  IF_UNIX = c$; % comment these lines out for Unix
  IF_DOS  = ;   % comment these lines out for Unix
%  IF_UNIX = ;  % comment these lines out for DOS
%  IF_DOS  = c$; % comment these lines out for DOS
%end_define
IF_UNIX [s,w] = UNIX('!rm dir1/dir2/file');
IF_DOS  eval('!del dir1\dir2\file.exe');
```

   When using many of these conditionals to replace MATLAB 'if... else' clauses, the resulting program will be smaller and faster.

2. MATLAB will display the results of an expression by removing the semicolon at the end of the expression. However, finding these missing semicolons at clean up time can be difficult. If debug printout needs to be re-enabled, an editing session is in order.

   A better way is:

```
%define
  BugPr = s$;% disables the debug printout (BugPr = ;)
  %BugPr = ; % enables the debug printout (BugPr = space)
%end_define;
a = foobar(barfoo(j.*k)) BugPr
```

   The debug printout can be enabled by switching which line within the define is commented out.

   Since a semicolon normally terminates a literal assignment within a define section, the s$ semicolon symbol can be used to include a semicolon within the assignment text.

3. As noted previously, it does not attempt to substitute a literal with its replacement text if the literal occurs within a quoted

string. Such behaviour would be dangerous and would destroy the ability to define arbitrary strings as needed; however sometimes it is useful to be able to make such replacements. To allow the replacements, use $$ in place of the quote character. MPP will replace the $$ with a single quote, as well as making any replacements for all literals found within the quotes. The $$ can also be used to define replacement text that has an odd number of quotes. This is rarely useful.

As explained previously, the $ character may be used in a literal only as the first or last character of the literal. The use of this special property of $ can be seen in the following example.

```
%define
  FileName$  = foo;
  Extention$ = .ini;
  In_file    = $$$$FileName$Extention$$$$$;
%                  file name, double quoted
  Permission = $$'r+'$$; % open for reading & writing
  ACT_f = 'Callback',$$fid = fopen(In_file,Permission)$$;
%end_define
set(Handle,ACT_f);
```

The output of mpp corresponding to this last line is:

```
set(Handle,'Callback','fid = fopen(''foo.ini'',''r+'')');
```

🕮 NOTE

If literals were used in FileName and Extention without the final $, then MPP would try to look up the literal FileNameExtention which would not be found.

Another special case which applies to define sections (or include files) is the specification of a literal with no definition. In this case, MPP will assign a value to this literal equal to one plus the value of the previously defined numeric literal. The syntax of this is best described by example.

Handle index definitions are usually made sequentially and can be created using a define section such as:

```
%define
  FIG    = 1;  % begin assigning handle index definitions
  RUN    = FIG+1;   % run button
  STOP   = RUN+1;   % stop button
  SLIDEV = STOP+1;  % voltage slider
  AXIS1  = SLIDEV+1;% axis 1
```

```
  AXIS2  = AXIS1+1; % axis 2
  AXIS3  = AXIS2+1; % axis 3
  PLOT1  = AXIS3+1; % plot 1
  PLOT2  = PLOT1+1; % plot 2
  PLOT3  = PLOT2+1; % plot 3
%end_define
```

However, using the special case described above, the define code can be replaced with the following source:

```
%define
  FIG = 1;           % begin assigning handle definitions
  RUN, STOP;         % run, stop buttons
  SLIDEV;            % voltage slider
  AXIS1,AXIS2,AXIS3; % axis 1,2,3
  PLOT1,PLOT2,PLOT3; % plot 1,2,3
%end_define
```

🕮 NOTE

If the auto-incrementing defines are used (as above), any of the pre-defined literals, or any literal containing the $ character, then the pre-processor source may not be run by MATLAB directly. Thus once any of these features are used, MPP must always be used on the source code. Even without using these special features, there are many literal assignments that will be legal and useful for MPP that are illegal when used as input to MATLAB directly. Once resigned to always running the source through MPP, these problems become moot.

There are many ways to use MPP to make your code clearer and easier to modify. A few of these ideas can be found in the following example.

Invoking MPP

Some editors allow automatic "compilation" of the source using MPP before exiting. This will make it less likely to accidently skip the MPP processing before running the output M-file. Locating a Windows based editor which supports this type of source processing is essential to getting the most from MPP. Two good shareware Windows editors are:

1. *WinEdit*
2. *E! for Windows*

MPP can also be invoked from a make file, which might look something like:

```
.SUFFIXES:
.SUFFIXES: .m .mi
.mi.m:
        mpp $*.mi
OBJS = program1.m program2.m
all: $(OBJS)
program1.m: program1.inc
program2.m: program2.inc
```

A MPP GUI Example

The previous examples provide an idea of the mechanics of the preprocessor. The example file: ***mpp_ex1.mi*** in the ***vex*** subdirectory demonstrates the use of MPP to achieve the benefits mentioned in the synopsis.

Before studying the code, it is instructive to run it once to see what it does. If MPP is not on the DOS path, you will have to type “path\mpp mpp_ex1.” Create ***mpp_ex1.m*** by typing "***mpp mpp_ex1***" at the DOS prompt. Run it by typing "***mpp_ex1***" at the MATLAB command prompt.

Usually the .mi source can be run by MATLAB without using MPP by just renaming it to have a .m extension; however, in this GUI example some preprocessor constructs (e.g. the definitions of the GUI objects & properties) are used that MATLAB cannot handle directly. Therefore, ***mpp_ex1.mi*** cannot be run without first processing the source with MPP.

INDEX
PROGRAMMING GUIDE