CAPS PROCESS CONTROL ENGINE
MATLAB INTERFACE SPECIFICATION

Rev 9/28/09

I. Overview

MATLAB client modules communicate with the PCE process using a single MEX DLL, PCEmex.mexw32. C# clients use a wrapper that provides language-compatible access to the PCE application program interface (API). All client language platforms communicate with the PCE API using a lower-level PCE shared memory interface C library.

For C or C# clients, the API library is implemented in PCEfunc.c.

All MATLAB functionality is implemented in a single function, PCEmex, which takes the form

outparams = PCEmex(func_code, step_id, param_id, max_elements, param_value);

-func_code is a symbol for a number which identifies the type of action PCEMEX will perform. It is always required.

- step_id is a symbol for a number which identifies the processing step within the PCE loop for which the action is done. It is required for some function codes (see Function Codes).

- param_id is a character string which identifies the specific PCE behavior variable name or MATLAB slave engine input or output variable name, or control subfunction, depending on function_code. It is required for some function codes (see Function Codes).

- max_elements is only used for functions PCEF_SETINVAR and PCEF_SETOUTVAR, and only on the first call for a given variable to create it and set its initial value. It gives the maximum element size (for a matrix, the number of cells, for a string, the number of characters) that that variable will hold now or in the future. This value will be saved and used to validate future calls to set that variable. If there is not enough space allocated in the first call to accommodate the current call's variable size, the current call is rejected.

- param_value is either a character string or a MATLAB matrix of double or unsigned-int-16 values associated with param_id. It is required for some function codes (see Function Codes).

II. Function Codes

function_code values should use a common name defined and used both within MATLAB and the PCE. For clarity, THE SYMBOLS RATHER THAN THE NUMERIC VALUES ARE TO BE USED IN THE ACTUAL CODING OF PCEmex CALLS.

Note: In the list below, slave refers to a slave MATLAB process function step

PCEF_SET_DEFAULT 1 // Set/reset PCE default parameter values

```
PCEF_CONFIGURE          2     // Configure any hardware before starting engine
PCEF_START              3     // Start PCE loop execution
PCEF_STOP               4     // Stop PCE loop execution
PCEF_TERMINATE          5     // Terminate the PCE process
PCEF_SET_IN_VAR         6     // Set a new or existing input variable to PCE/slave
PCEF_SET_OUT_VAR        7     // Identify variable to be picked up from slave step
PCEF_GET_VAR            8     // Get the value and capacity of a PCE variable
PCEF_SET_CONTROL        9     // Alter the control flow for a PCE processing step
PCEF_GET_CONTROL        10    // Obtain control setting for a PCE processing step
PCEF_GET_VAR_LIST       11    // Returns a comma-delimited list of slave/all variable ids
PCEF_WAIT_VAR_VALUE     12    // Wait for a dbl scalar variable to reach/exceed a value
PCEF_GET_STEP_IDS       13    // Returns a list of all step id and step names
PCEF_TRAIN_CLASFR       14    // Trains a pattern recognition classifier
PCEF_TEST_CLASFR        15    // Tests a pattern recognition classifier
PCEF_TERMINATE_ALL      16    // Terminate BOTH the PCE and DAQ process
PCEF_PCE_LOGGING_OFF    17    // Turn off PCE logging of shared memory changes
PCEF_PCE_LOGGING_ON     18    // Turn on PCE logging of shared memory changes
PCEF_PCE_DVTRAIN_CLASFR 19 // Train all classifiers using device-prompting
PCEF_PCE_DVTRAIN_CLASFR_CAN 20 // Cancel train classifiers using device-prompting
```

III. Function Code Call Syntax for Each Code

```
PCEmex(PCEF_SET_DEFAULT, <optional-sec-timeout>);
                        // Set/reset PCE default parameter values
                        // times out if not completed in <optional-sec-timeout> secs
                        //      defaults to 60 seconds

PCEmex(PCEF_CONFIGURE, <optional-sec-timeout>);
                        // Configure any hardware before starting engine
                        // times out if not completed in <optional-sec-timeout> secs
                        //      defaults to 60 seconds

PCEmex(PCEF_START, <optional-sec-timeout>);
                        // Start PCE loop execution
                        // times out if not completed in <optional-sec-timeout> secs
                        //      defaults to 60 seconds

PCEmex(PCEF_STOP, <optional-sec-timeout>);
                        // Stop PCE loop execution
                        // times out if not completed in <optional-sec-timeout> secs
                        //      defaults to 60 seconds

PCEmex(PCEF_TERMINATE, <optional-sec-timeout>);
                        // Terminate the PCE process
                        // times out if not completed in <optional-sec-timeout> secs
                        //      defaults to 60 seconds
```

For the following functions, see step_id, param_id, and param_value sections below.

```
PCEmex(PCEF_SET_IN_VAR, step_id, param_id, [param-value],[ maxelements]);
                                        // Set a new or existing input variable to PCE/slave
                                        // If param-value present, sets variable to that value
                                        // If step_id not 0, used to specify variable as input
                                        // to MATLAB slave process.
                                        // If maxelements present, max size in elements
                                        //        for matrices, max number of cells
                                        //        for strings, max number of characters
                                        // maxelements is used only on the first call for a
                                        // given variable to allocate shared memory for it;
                                        // any future calls do not require this parameter, but
                                        // are edited against maxelements to ensure that the
                                        // new variable data will fit in the same space.

PCEmex(PCEF_SET_OUT_VAR, step_id, param_id, [param-value],[ maxelements]);
                                        // If param-value present, sets variable to that value
                                        // If step_id not 0, used to specify variable as output
                                        // from MATLAB slave process.
                                        // if maxelements present,  max size in elements
                                        //        for matrices, max number of cells
                                        //        for strings, max number of characters
                                        // maxelements is used only on the first call for a
                                        // given variable to allocate shared memory for it;
                                        // any future calls do not require this parameter, but
                                        // are edited against maxelements to ensure that the
                                        // new variable data will fit in the same space.

[outparam, optional-max-elem] = PCEmex(PCEF_GET_VAR, param_id);
                                        // Retrieves the value of the identified variable
                                        // into outparam; if 2nd outparm present, loads it
                                        // with maximum elements (matrix cells, string chars,
                                        // etc) allowed/allocated for the variable

PCEmex(PCEF_SET_CONTROL, step_id, param_id[,optional_param_value],[optional_mode]);
                                        // Sets the processing control flow for PCE step
                                        // param_id possible values for step's cntl func:
                                        // "ADDBEFORE" executes slave before PCE step
                                        // "ADDAFTER" executes slave after PCE step
                                        // "REPLACE" executes slave instead of PCE step
                                        // "BYPASS" bypasses PCE step only
                                        // "ENDBEFORE" does not process this or later steps in
                                        //                        this loop
                                        // "ENDAFTER" processes no later steps in this loop
```

```
                                        // "NONE" PCE executes in normal default mode
                                        // "RESET" forces control to "NONE" regardless of
                                        //                 prior state
                                        // param value is MATLAB cmd string to execute
                                        //
                                        // NOTE!!: Once a new slave control of "ADDBEFORE",
                                        // "ADDAFTER", or "REPLACE" is set, it cannot
                                        // be changed except with a "RESET" command.
                                        // mode defines the run mode for the slave.  legal values:
                                        // 1 = MATLAB script

[outcntl outcmd mode] = PCEmex(PCEF_GET_CONTROL, step_id);
                                        // outcntl returns value of step's control function,
                                        //         eg "REPLACE"
                                        // outcmd returns the MATLAB command string
                                        //         associated with the control function
                                        // mode defines the run mode for the slave process
                                        // eg 1 = MATLAB script

varlist = PCEmex(PCEF_GET_VAR_LIST,[optional-step-id, io-spec]);
                                        // returns a string containing a comma-separated
                                        // list of all slave variables for that step, and
                                        // io-spec (1=input, 2=output), - or-
                                        // list of all variable ids if step-id absent or 0

actual_value = PCEmex(PCEF_WAIT_VAR_VALUE, var-id, wait_value, <optl-sec-timeout>)
                                        // waits for either (a) dbl variable with id var-id to
                                        // reach or exceed the value wait_value, or (b) for the PCE
                                        // to be in stopped (not looping) state, then returns the
                                        // variable value to actual_value; times out if not completed
                                        // in <optl-sec-timeout> seconds: defaults to 60 seconds

 [outids outnames] = PCEmex(PCEF_GET_STEP_IDS)
                                        // returns a vector containing step ids
                                        // col = 1   row = number of steps
                                        // and another vector containing step names
                                        // col = max name length, row = number of steps

PCEmex(PCEF_TRAIN_CLASFR);  // Start multiple-loop PCE runs to train PR classifier

PCEmex(PCEF_TEST_CLASFR);     // Start multiple-loop PCE runs to test a PR classifier

PCEmex(PCEF_PCE_LOGGING_OFF); // Turn off PCE logging of shared memory changes

PCEmex(PCEF_PCE_LOGGING_ON);  // Turn on PCE logging of shared memory changes

PCEmex(PCEF_PCE_DVTRAIN_CLASFR);  // Train all classifiers using device-prompting
```

PCEmex(PCEF_PCE_DVTRAIN_CLASFR_CAN); // Cancel device-prompted classifier train

IV. Step IDs

The step_id identifies the PCE processing step in its default processing loop to which a parameter refers. It is REQUIRED for function codes

```
PCEF_SET_IN_VAR        6     // Set new/existing input variable to PCE/slave step
PCEF_SET_OUT_VAR       7     // Identify variable to be picked up from slave step
PCEF_SET_CONTROL       9     // Alter the control flow for a PCE processing step
PCEF_GET_CONTROL       10    // Obtain control setting for a PCE processing step
```

The step_id parameter is absent for function code PCEF_GET_VAR, because variable names are unique across all steps of the PCE.

Step id possible values are

```
PCES_NORM              0     // No specific step; used with PCED_SET_*_VAR
                             // to indicate that this variable is not to be delivered to or
                             // picked up from any step's MATLAB slave process.
PCES_DAQ               10    // Data acquistion
PCES_BP_FILTER         20    // Signal Conditioning Bandpass Filter (10-500Hz)
PCES_NOTCH_FILTER      30    // 60 Hz Notch Filter
PCES_ECG_CLIP          40    // ECG Clipping
PCES_HP_FILTER         50    // High-pass filtering
PCES_MAV               60    // Mean Absolute Value Calculation
PCES_AVG_CHAN_POWER    70    // Pattern Recognition Power Avg Across Channels
PCES_FEAT_EXTRACT      80    // Pattern Recognition Feature Extraction
PCES_CLASSIFY          90    // Pattern Recognition Classify
PCES_PR_MVOTE          100   // Pattern Recognition Majority vote
PCES_CHAN_MAV_MRG      105   // Merge physical channel MAV and classifiers' class MAV
PCES_XFR_FUNCTION      110   // Map in data channels to out control channels
PCES_STATE_MACHINE     115   // Apply optional state-machine logic
PCES_GS_FILTER         120   // Grasp State Filter.. Modfies DOF based on hand logic
PCES_XFR_FILTER        130   // Implementat'n-specific control output adjustments
PCES_VR_OUT            135   // Presents filtered and speed-applied DOF signals to VR
PCES_MOTOR_MAP         140   // Map commands to drive motors / VR
PCES_MOTOR_GAIN        150   // Apply motor gain to arrive at final output
PCES_MOTOR_OUT         160   // Write motor signal to drive hardware
```

V. Param IDs

Param IDs are character strings that identify the specific PCE behavior variable name or MATLAB slave engine input or output variable name, or control subfunction, depending on function_code. These can be used to get or set an existing variable, or create and populate a new variable, or merely specify a variable for use by a MATLAB slave process, or specify a control operation. See the Step ID - Param ID Mapping Section for a description of parameter ids and their associated format of possible values and role as input/output to the various steps.

VI. Param Values

Must be a scalar double, a matrix of doubles or uint-16s, or a string, based on Param-ID. See next section for a description of Param IDs and their associated format of possible values and role as input/output to the various PCE steps.

VII. Default PCE Step-ID - Param ID - Param Value Mapping

Non-step-specific variables

General Variables:

```
PCE_STEPID              // scalar double, id of current step being executed
FRAME_CNT               // scalar double, number of frames completely processed
PCE_RUNNING             // scalar double, PCE State; 0 = stopped, else running
LOOP_ELAPSED_TIME       // scalar double, time elapsed since last frame read io
MIN_INTERLOOP_SLEEP_MS // min sleep time in ms between PCE frame reads
                        // used only if playing DAQ data back from a file
                        //  if -1, replay data to simulate hardware (~ frame incr rate)
```

Classifier Training Variables:

```
TRAIN_CLASFR_NUM        // classifier number for which to perform PR training
TRAIN_FILE_CLASS_NUM// class number of current file used for PR training
TRAIN_STATUS            // scalar double, pattern recognition training/testing status:
                        // 0 = Inactive, last operation (if any) successful
                        // 1 = Active (in-progress)
                        // 2 = Aborted (error while training/testing, now Inactive)
                        // 3 = Active testing (in-progress)
TRAIN_PCT_DONE          // scalar double, PR training/testing approx % complete
TRAIN_DATA_FOLDER       // root folder containing raw data files for training/testing
                        // file names are Cccc_yymmdd_hhmmss_hhmmss.DAQ
                        // ccc = actual (target) class recorded
                        // yymmdd_hhmmss is the date_time classes were recorded
                        // the 2nd hhmmss is the creation time of this specific file
TRAIN_FEAT_FOLDER       // if not null, folder into which train/test feature data written
REDUC_FEAT              // Reduced train feat frames after ULDA_G mult
```

```
                                // This output only available in current PCE process run,
                                // following training of classifier; used by GUI for View
                                // (rows=frames, cols=3)
        REDUC_CLASS             // Actual class id of each frame of REDUC_FEAT
                                // This output only available in current PCE process run,
                                // following training of classifier; used by GUI for View
                                // (rows=frames, cols=1)

        Device-Prompted Data Capture / Training Variables:

        DVTRAIN_STATUS          // scalar, device-prompted train state, set by PCE
                                // 0=Idle, last dvtrain operation (if any) successful
                                // 1=Reset, Initial device position moving to endpoint
                                // 2=Reset, device position moving from endpoint to neutral
                                // 3=Data Collection in progress
                                // 4=Classifier training in progress
                                // 5=Last Retrain aborted due to error
        DVTRAIN_CAPT_DELAY      // scalar, dev. Train delay between captures, set by GUI
        DVTRAIN_CAPT_DURATION   // scalar, train duration of each capture, set by GUI
        DVTRAIN_ITERATIONS      // scalar, # iterations of each class to capture, set by GUI
        DVTRAIN_DOF_SPEED       // dev train data collect prompt speed for DOFs, set by GUI
                                // rows=DOFs, cols=1
        DVTRAIN_COLLECT_CLASS   // scalar, current data-collection class, set by PCE

        Slave Process Variables:

        SLAVE_STATUS            // scalar double, code retnd by slave process; non-0=error
        SLAVE_MSG               // string, message retnd by slave process
        PCE_SLAVE_PAUSE_STEP// step id of PCE step where PCE paused waiting for slave;
                                // slave process resumes PCE by setting this variable to 0

PCES_DAQ                10      // Data acquistion
        Input Param IDs and Values (can be set only when PCE stopped) :
                DAQ_IN_FNAME=string input filename to use as raw data source
                        PCE default = "" (use live hardware inputs)
                DAQ_BOARD_TYPE=scalar double DAQ board/interface type
                        1 = MCC USB-1616FS or equivalent
                        2 = Otto Bock Serial Interface
                        PCE default = -1
                DAQ_BOARD=scalar double base DAQ board number channels 1-16
                        PCE default = 1
                DAQ_SAMP=scalar double sampling rate (datapoint each channel per sec)
                        PCE default = 1000
                DAQ_FRAME=scalar double frame size
                        PCE default = 150
                DAQ_FRINC=scalar double frame increment, in datapoints for each chan
```

PCE default = 100
USE_FILE_FRAMING=scalar double, if inputting data from a file,
0=use frame size, incr from user's current settings
1=use frame size, incr from file header at file record time
DAQ_OUT_FNAME=string output filename recording raw data
PCE default = "" (no file output)
DAQ_CHAN = 2 row matrix of doubles, row 1 = channel #, row 2 = gain spec
PCE default = 16 cols, row 1=1-16, row 2 = zeros
gain-spec default = BIP5VOLTS (-5V to 5V)
DAQ_FILE_DATAPTS = scalar double, for file inputs, the number of samples taken for each channel
DAQ_FILE_MAV = row vector of uint-16s, cols = same as for DAQ_CHAN, the mean absolute value of each active channel for all file datapoints

Output Param IDs and Values:
DAQ_DATA=matrix of uint-16s, rows=channels, cols=DAQ_FRAME
PCE default = 16 rows, columns = 150
PCE maxelements = maxchannels* MAXFRAMESIZE = 16000

PCES_BP_FILTER 20 // Signal Conditioning Bandpass Filter
PCE default control = BYPASS
Input Param IDs and Values:
DAQ_DATA
FILTER_CHAN=vector of doubles, bitmask for applying various chan filtering
bit value of 1=filter, 0=nofilter, rows=channels, cols=1
PCE default = 16 rows, columns = 1
PCE maxelements = maxchannels
Bitmask values (logically OR'd together in matrix element)
1 = PCEFILTER_BP = enable BP filtering
2 = PCEFILTER_NOTCH = enable Notch filtering
4 = PCEFILTER_ECG = enable ECG_CLIP processing
8 = PCEFILTER_HP = enable HP filtering
16 = PCEFILTER_MAV_FRAME = calculate MAV over full frame rather than just frame increment
BP_LO_CUT=scalar double (Low cutoff freq)
PCE default = 10
BP_LO_ORD=scalar double (Low filter order)
PCE default = 3
BP_HI_CUT=scalar double (High cutoff freq)
PCE default = 500
BP_HI_ORD=scalar double (High filter order)
PCE default = 3
Output Param IDs and Values:
BP_DATA = matrix of doubles, rows= DAQ_FRAME, cols=channels
PCE default = 150 rows, columns = 16
PCE maxelements = maxchannels* MAXFRAMESIZE = 16000

PCES_NOTCH_FILTER 30 // Signal Conditioning Bandpass Filter

PCE default control = BYPASS
Input Param IDs and Values:
BP_DATA
FILTER_CHAN
NOTCH_FREQ=dbl vector of center frequencies of notches
rows = 1, columns = list of frequencies (0=no more values)
PCE default = 60, 180, 300 (3 entries)
PCE maxelements = 10
NOTCH_Q=Q value of notch filter
PCE default = 35???
Output Param IDs and Values:
NOTCH_DATA = matrix of doubles, rows= DAQ_FRAME, cols=channels
PCE default = 150 rows, columns = 16
PCE maxelements = maxchannels* MAXFRAMESIZE = 16000

PCES_ECG_CLIP 40 // ECG Clipping
PCE default control = BYPASS
Input Param IDs and Values:
NOTCH_DATA
FILTER_CHAN
ECG_FLEN=scalar double (ECG clip framelen)
PCE default = 200
ECG_GAIN=scalar double (ECG clip gain factor)
PCE default = 2.1
Output Param IDs and Values:
ECG_DATA= matrix of doubles, rows= DAQ_FRAME, cols=channels
PCE default = 150 rows, columns = 16
PCE maxelements = maxchannels* MAXFRAMESIZE = 16000

PCES_HP_FILTER 50 // High-pass filtering
PCE default control = BYPASS
Input Param IDs and Values:
ECG_DATA
FILTER_CHAN
HP_CUT=scalar double (HP cutoff freq)
PCE default = 70
HP_ORD=scalar double (HP filter order)
PCE default = 3
Output Param IDs and Values:
HP_DATA= matrix of doubles, rows= DAQ_FRAME, cols=channels
PCE default = 150 rows, columns = 16
PCE maxelements = maxchannels* MAXFRAMESIZE = 16000

PCES_MAV 60 // Mean Absolute Value Calculation
Input Param IDs and Values:
HP_DATA

Output Param IDs and Values:
CHAN_MAV=vector of doubles, rows=channels, cols=1

PCES_AVG_CHAN_POWER 70 // Pattern Recognition Power Avg Across Channels
Input Param IDs and Values:
CHAN_MAV
CLASFR_CHAN=matrix of 0/1 doubles, classifiers and channels used for each,
rows=channels, cols=classifiers
(supports multiple classifiers)
Set by MATLAB maxelements = maxchannels * 6 = 192
Output Param IDs and Values:
CLASFR_MAV=vector of MAV scalar doubles, rows=1, cols=classifiers

PCES_FEAT_EXTRACT 80 // Pattern Recognition Feature Extraction
Input Param IDs and Values:
HP_DATA
CLASFR_CHAN

FEAT_SELECT=vector of doubles, selects features to extract for each classifier.
rows = 1, cols = classifiers
This is in effect a bitmap that allows one or more features to be
selected, including:
1=Mean Relative Abs Value (MAV after normalizing to data
mean value as zero point - original "MAV" used in ACE)
2=Waveform Vertical Length
4=Zero Crossings
8=Slope Changes
16=Mean Absolute Value (not normalized to data mean; new
feature as of 5/30/08)
32=Aautoregressive features (number = ARorder, dflt = 6)
PCE default value = 15 (selects 4 features used in ACE)
Set by PCE maxelements = maxclassifiers = 6

FEAT_OUT_FNAME1,2..N=string output filename recording features
Note: output data is concatenated - file should be set empty or
be nonexistent for first data set collected.
Note: file opened at PCE start
PCE default = "" (no file output)

Output Param IDs and Values:
FEAT_DATA1, 2..N=vector of doubles, rows=1, cols=features
Set by MATLAB maxelements = FeatTypes * maxchannels = 128
(variable suffix supports multiple classifiers)

PCES_CLASSIFY 90 // Pattern Recognition Classify
Input Param IDs and Values:

FEAT_DATA1, 2..N
FEAT_MEANS1, 2,…N=vector of doubles, rows=classes, cols=features
Set by Client maxelements = 25*64
ADJR1, 2,…N=matrix of doubles, rows=features, cols=features
Set by Client maxelements = 64*64
(variable suffix supports multiple classifiers)
ULDA_G1, 2,…N=matrix of doubles, rows=features, cols=classes
` ULDA transformation matrix (not currently used)
Set by Client maxelements = 64*16
(variable suffix supports multiple classifiers)
CLASFR_CLAS1, 2,…N=actual class id numbers associated with given classifier, order corresponding to that in FEAT_MEANSn rows
vector of doubles, rows = classes, cols = 1
(variable suffix supports multiple classifiers)

Output Param IDs and Values:
CLAS_OUT=actual class id number output of classifier, eg wrist flexion, etc
vector of doubles, rows=1, cols=classifiers
ENTROPY= entropy associated with classification decision,
vector of doubles, rows=1, cols=classifiers
TEST_CONFUSION_MTRX1, 2,..N=matrix of doubles, classification test confusion matrix; each cell records number of combinations of specific instances of actual and predicted class.
rows=cols= # clasfr classes; rows=actual, cols=predicted
Matrix is updated only when replaying data (see variable DAQ_IN_FNAME) where filename is in form Cccc_...,
where ccc is the actual class number (see variable TRAIN_DATA_FOLDER for file name conventions)
(variable suffix supports multiple classifiers)

PCES_PR_MVOTE 100 // Pattern Recognition Majority vote
Input Param IDs and Values:
CLAS_OUT
PR_MV_WINDOW = matrix of dbls, rows = num PR votes, cols = classifiers
Set by MATLAB maxelements = 50 * 6 ??
Output Param IDs and Values:
MV_CLAS_OUT=class id number vector of doubles, rows=1, cols=classifiers
CLAS_OUT vector filtered by majority vote

PCES_CHAN_MAV_MRG 105 // Merge physical channel MAV and classifier's class MAV
// into single LOG_CHAN_MAV
Input Param IDs and Values:
CHAN_MAV
CLASFR_MAV
MV_CLAS_OUT
Output Param IDs and Values:
LOGIC_CHAN_MAV=logical channel MAV values

rows=maxchannels(default=16)+maxclasses(default 50),
cols=1
for 1st maxchannel rows, identical to CHAN_MAV
for final maxclasses rows, the CLASFR_MAV value
placed in row slot indexed by class id number

PCES_XFR_FUNCTION 110 // Map in data channels to out control channels
Input Param IDs and Values:
LOGIC_CHAN_MAV
DOF_ID_MAP = map of active DOF columns in DOF_ACT to DOF ID numbers
rows=1 cols=active DOFs as in DOF_ACT.
Cells contain DOF ID for corresponding DOF_ACT slot.
Set by GUI Client.
Note: not used by PCE; passed thru for use by VR process
XFR_CNTL=matrix of doubles, rows=CntlFields, cols=# DC Classifiers (DCC)
Set by Client maxelements = DC Classifiers * CntlFields
CntlFields:
controlMode = 0; // 0: Unconfigured1: Single Site, 2: Multi-Site
activationMode = 1; // 1:6, form of conventional control
// PCEAM_NO_CONFIG=0
// PCEAM_SGLSITE_DBLTHRSH=1
// PCEAM_SGLSITE_BIDIREC=2
// PCEAM_SGLSITE_RATESENS=3
// PCEAM_MULTISITE_DIFFER=4
// PCEAM_MULTISITE_FRSTCOME=5
// PCEAM_MULTISITE_MAX=6
isPorportional = 1; // 0 forces output to be binary on/off
onThreshToArmTime;// time to wait before determining if switch movemt
onThreshToTrigTime;// Used for switching & sgl site, rate sensitive cntl;
// time limit for activation
selectedClass; // currently selected class for this DCC
// ordinal class idx with XFR_CNTL struct
numXfrSets; // Number of transfer set entries:
// for switches: 1, for DCC: 8
xfrSet[numXfrSets] // Cntls both sides of 1-8 Xfr sets; each has rows:
actColIdx; // The DOF_ACT or SWITCH_ACT
// column idx, starts with 0 (-1 = not used)
chan1; // 1st input chan # starting with 1
chan2; // 2nd input chan # starting with 1
onThresh1; // Input 1 threshold; switch-on thresh.
onThresh2; // Input 2 threshold; switch-on thresh.
armThresh1; // Input 1 switch-arm thresh.
armThresh2; // Input 2 switch-arm thresh.
inputGain1; // Gain to be applied to input 1
inputGain2; // Gain to be applied to input 2
outputGain; // Gain to be applied at output

SWITCH_CNTL= switch activation functions; format same as XFR_CNTL except 1 xfrSet entry instead of 8.
matrix of doubles, rows=CntlFields, cols= # Switch Events (SE)
Set by Client maxelements = SwitchEvents * CntlFields

SWITCH_NODE = switching endpoint node defs: DOF or parent of DOF group
matrix of doubles, rows=CntlFields, cols = # nodes
Set by Client maxelements = 50 * CntlFields

CntlFields:

```
DOF_ACT col #       // The DOF_ID_MAP or DOF_ACT
                    // column idx, starts with 0 (-1 = group, not a DOF)
currentChild;       // idx of active child of this node (if a DOF, -1)
parent;             // idx of parent of this node (if highest node, -1)
```

SWITCH_EVENT = switching event (SE) designated by labelled arrow on GUI
matrix of doubles, rows=CntlFields, cols = # switch events (SEs)
Set by Client maxelements = 50 * CntlFields

CntlFields:

```
class id;           // dynamic switch event class id associated w/ SE
holdTime;           // time (ms) from initial triggering before next
                    // trigger permitted
```

SWITCH_MAP = state transition map cells give next active switch child node for a given switch node (SWITCH_MAP col idx = leaf-DOF SWITCH_NODE col idx) when a given SWITCH_EVENT (SWITCH_MAP row idx = SWITCH_EVENT col idx) occurs. This matrix contains all switching networks defined, even if they are not connected (and therefore run in parallel).
See DCClassifierSwitching.doc section on Switch Transition Processing for details on PCE usage.
mtrx of dbls, rows=Switch Events, cols=# DOF-type switch nodes
Set by Client maxelements = 50 * # active DOF_ACT cols (25)

Output Param IDs and Values:

DOF_ID_MAP

DOF_ACT= matrix of doubles, rows=CntlOutFields, cols=DOFs
Set by MATLAB maxelements = DOFs * CntlOutFields

CntlOutField Rows:

```
1=dofAction;        // DOF output action, expressed as % of full speed,
                    // + or - indicating direction (range -100 to +100)
2=gainedInput1;     // input1 * gain1
3=gainedInput2;     // input2 * gain2
4=adjVal1;          // Single: max(0, input1 * gain1 – threshold1)
5=adjVal2;          // Single: max(0, input1 * gain1 – threshold2) or
                    // Double: max(0, input2 * gain2 – threshold2)
```

6=rsState; // rate sensitive mode; current switching status
7=rsElapsed; // rate sensitive elapsed time / elapsed time in
// current switch status
8=max1 = 0; // Maintains max gainedInput1 value
9=max2 = 0; // Maintains max gainedInput2 value

PCES_STATE_MACHINE 115 // Apply optional state-machine logic
Input Param IDs and Values:
DOF_ACT
SM_STATE = mtrx of dbls, current machine state (rows=fields max 53, cols = 1)
SM_STATE rows:
1 = state machine node id in SM_NODE matrix
2 = processing function code for this state
3 = number of variable parameters up to max of 50
4+ = list of parameters
SM_NODE = mtrx of dbls, set of all possible state nodes
(rows=fields max 56, cols=#-nodes max 50)
SM_NODE rows:
1 = state machine node id
2 = reserved
3 = parent node id for collection of state nodes (-1 = no parent)
4 = currently active child node id of this parent node (-1 = no child nodes)
5 = processing function code for this state
6 = number of variable parameters up to max of 50
7+ = list of parameters
SM_EVENT = mtrx of dbls, set of all possible events
(rows=fields max 54, cols=#-events max 50)
SM_NODE rows:
1 = event id
2 = reserved
3 = processing function code for determining whether event fired
4 = number of variable parameters up to max of 50
5+ = list of parameters
SM_SWITCH_MAP = dbl mtrx, state transitions tbl for current state, fire event
Cell contents = SM_NODE id of next state (-1=no chg)
(rows=events max 50, cols=nodes max 50)
SM_EVENT_FIRED = dbls vector indicating whether event fired, 1=yes, 0=no
(rows=1, cols=events max 50)
SM_HOLD_TIME = scalar dbl, time in ms req'd after event firing before next fire
Output Param IDs and Values:
SM_STATE
SM_DOF_ACT = vector of doubles, rows=1, cols=DOFs (Modified DOF)

PCES_GS_FILTER 120 // Grasp State Filter, Modifies DOF based on hand logic
PCE default control = BYPASS
Input Param IDs and Values:

SM_DOF_ACT
DOF_HAND_GRASP = vector of 0's or 1's where the given DOF is a hand grasp
Or "No movement" (rows = 1, cols = DOFs)
GSF_MV_WINDOW = vector of doubles, rows = num GS votes, cols = 1
Contains 0, or the DOF id value of the hand grasp
Set by Client maxelements = 50
GSF_STATE = scalar double, Current grasp state. This is the grip's DOF_ID value.
PCE default 0 (Not Defined)
GSF_HAND_POS = scalar double, Percent Hand Open
PCE default 100 (Hand is open) passed back from VR
this variable updated by VR, passed thru PCE without alteration
VR_POS_OUT=dbl vec, VR actual DOF position in degrees set by VR,
(rows=1, cols=DOFs)
this variable updated by VR, passed thru PCE without alteration
GSF_THRESH_POS = scalar double, Percent open at which grasp select is allowed, PCE default 100 (Fully Open)

Output Param IDs and Values:
GSF_DOF_ACT = vector of doubles, rows=1, cols=DOFs (Modified DOF)

PCES_XFR_FILTER 130 // Implementat'n-specific control output adjustments
PCE default control = BYPASS

Input Param IDs and Values:
GSF_DOF_ACT
FDOF_ACT_PREV=vector of dbls, rows=1, cols= DOFs (previous FDOF_ACT)
XFR_FILT_SELECT=vector of doubles, rows=1, cols=DOFs
Cell value selects filters to use for that DOF. This value is in effect a bitmap that allows one or more filters to be applied:
1=Exponential function transforms invalue to outvalue using variables EXP_FILT_ECOEF and EXP_FILT_EMULT (each with rows=1, cols=DOFs) where
outvalue=
EXP_FILT_ECOEF * (e^(invalue* EXP_FILT_EMULT))

Output Param IDs and Values:
FDOF_ACT=vector of doubles, rows=1, cols= DOFs (filtered GSF_DOF_ACT)
FDOF_ACT_PREV (updated)

PCES_VR_OUT 135 // Presents filtered and speed-applied DOF signals to VR
// Controls signal driven figure and optional manually-
// driven ghost figure

Input Param IDs and Values:
FDOF_ACT
VR_ENABLE=scalar dbl, 0=disable, 1=enable VR output
VR_NORMALIZE=scalar dbl, 0=disable, 1=enable VR speed normalization
VR_GAIN=vec of dbls, gain applied to FDOF_ACT, (rows=1, cols=configDOFs)
VR_POS_IN=dbl mtrx, VR set position override, (rows=1 or 2, cols=all DOFs)

First row = DOF posititons for signal VR image
Opt. Second row = DOF positions for non-signal ghost VR image
Position given as percent of full range from center, -100 to +100
<-100 or >100 = no override, -100<=X<=100 = position to set
this variable from GUI/client passed thru to VR without alteration

VR_GHOST_COLOR=vec of dbls, VR ghost image color, (rows=1, cols=3)
1st column = red value
2nd column = green value
3rd column = blue value
0 <= value <= 255; 0 = zero intensity, 255=highest intensity
-1 = default/no change, >=0 = intensity to set
this variable from GUI/client passed thru to VR without alteration

VR_GHOST_TRANSPARENCY=scalar dbl, VR ghost image transparency
0 <= value <= 255; 0 = invisible, 255=opaque
-1 = default/no change, >=0 = degree position to set
this variable from GUI/client passed thru to VR without alteration

VR_CAMERA=dbl mtrx, sets VR camera position and angle
(rows=6, cols=1)
First three rows = camera X, Y, Z position
-1000 <= value <= 1000, < -1000 = no action
Next three rows = camera X, Y, Z angle
-180 <= value <= 180, < -180 = no action
(these values set from fields in OutDevice Config)
this variable from GUI/client passed thru to VR without alteration

VR_SIDE_IN=scalar dbl, set vr side, 1=right, 2=left.
VR will reset variable back to 0 if it was able to apply the changes,
-1 if failed due to video card compatiblities.

Output Param IDs and Values:

VR_FRAME_CNT=scalar dbl, updated by the VR to keep the GUI in sync.

VR_HWND=String representation of HWND value to VR window. Allow
External processes to control the VR's window.

VR_ACT=dbl vec, VR output speed signals (rows=images, cols=configDOFs)
First row computed as cell multiply FDOF_ACT * VR_GAIN
Second row programmatically set for ghost image, passed thru

VR_POS_OUT=dbl mtrx, VR actual DOF position in degrees set by VR,
(rows=2, cols=all DOFs)
First row = DOF posititons for signal VR image
Opt. Second row = DOF positions for non-signal ghost VR image
Position given as percent of full range from center, -100 to +100
<-100 or >100 = no override, -100<=X<=100 = position to set
this variable from GUI/client passed thru to VR without alteration

PCES_MOTOR_MAP 140 // Map output commands to output drive motors
PCE default control = BYPASS

Input Param IDs and Values:

FDOF_ACT

MOTOR_MAP=matrix of doubles indicates motor used for this DOF
// (1=used, 0=not) rows=motors, cols=DOFs
// Set by Client maxelements = 20*20
MANUAL_MOTOR_CONTROL=scalar double flag, non-0 = force override
// manual speeds from MANUAL_MOTOR_SPEED vector
// into MOTOR_CMD output
MANUAL_MOTOR_SPEED= vector of dbls, rows=motors, cols=1, override
// values to MOTOR_CMD output vector if
// MANUAL_MOTOR_CONTROL flag is non-0

Output Param IDs and Values:

MOTOR_CMD=vector of dbls, rows=motors, cols=1 (motor raw logical cmds)
// expressed as % of full speed,
// + or - indicating direction (range -100 to +100)

PCES_MOTOR_GAIN 150 // Apply motor gain and scaling to give final D/A output

PCE default control = BYPASS

Input Param IDs and Values:

MOTOR_CMD
ALLOW_MOTOR_OUTPUT=scalar double,
// 0=disable all motors,
// 1=enable according to MOTOR_ENABLE
MOTOR_ENABLE=vector of doubles, 1=enable, 0=disable
// rows=motors, cols=1
MOTOR_DA_MAP = matrix of dbls, maps motor to associated D/A pin(s)
// rows=motors, cols=2
// 1st column gives output D/A chan if MOTOR_SIG < 0
// 2nd column gives output D/A chan if MOTOR_SIG >= 0
// Set by MATLAB maxelements = 20*2
DA_GAIN=vector of dbls, rows= DAQ board D/A channels, cols=1 (D/A gains)
// Set by MATLAB maxelements = 16
DA_OFFSET=vector of uint-16s, D/A offset = value added to gained and uint-16
// scaled MOTOR_CMD; can also be defined as the uint-16
// D/A output setting that corresponds to no movement
// rows= DAQ board D/A channels, cols=1
// Set by MATLAB maxelements = 16
DAQ_OUT_RANGE = dbl scalar, gain range spec for all out D/A chans
// (see DAQ_CHAN) output board (-1=disabled)
// also used at time of PCE Looping Start
// PCE assumed default: 100 = UNI10VOLTS (0 to +10V)
// Note: NOT PCE-programmable on USB-3114; param
// is ignored; range pre-setup by Instacal to one of:
// 100 = UNI10VOLTS (0 to +10V)
// 1 = BIP10VOLTS (-10V to +10V)

Output Param IDs and Values:

DA_SIG=vector of uint-16, rows=DAQ board D/A channels, cols=1
// data to be delivered in final form to output D/A

// Note default data also output at time of PCE Start
// Set by MATLAB maxelements = 16

PCES_MOTOR_OUT 160 // Write motor signal to drive D/A hardware

PCE default control = BYPASS

Input Param IDs and Values:

DA_SIG

DAQ_OUT_BOARD_TYPE = dbl scalar, D/A output board type (-1=disabled)
// 1=Measurement Computing Corp USB-3114
// 2=Otto Bock Serial Interface
// also used at time of PCE Looping Start

DAQ_OUT_BOARD = dbl scalar, board # of D/A output board (-1=disabled)
// also used at time of PCE Looping Start

Output Param IDs and Values:

None