CBPM File Input Overview

The CBPM data file input routines provide a standardized method of accessing collections of BPM data on disk and storing them in local application memory for use. A set of functions that form an application programming interface (API) for this mechanism are provided.

Languages supported

This mechanism is provided in two forms, depending on the programming language the developer uses to work with the data.

C & Fortran

A static library `libcbpmfio.a` is provided.
Additionally for Fortran, an f90 module is provided to declare the interface.

```
  cbpm_file_input.mod
```

Matlab

A shared object library `libcbpmfio.so` is provided along with a collection of <function>`.mexa64` files, one for each function available in the API.
A matlab ".m" file is provided to set everything up in a single step for the end user.

Setting up for use

Prerequisites

This data reading system is designed to be used on CLASSE computer systems only. For all supported languages, your BASH shell-compatible environment must be set up to contain the laboratory's ACC build system environment variables which contain important path information.
In order to do this one must invoke:

```
ACC_SETUP_SCRIPT=/nfs/acc/libs/util/acc_vars.sh
if ( [ -f ${ACC_SETUP_SCRIPT} ] ); then
    source ${ACC_SETUP_SCRIPT}
fi
```

A good place for this clause is in a login script such that upon spawning a new terminal, the user's environment is configured to use the default, development release of pre-built accelerator libraries and applications.

Building C and Fortran code using the ACC build system

To link against libcbpmfio, one must reference it in their makefile(s).

```
CESR_LIBS   := cbpmfio
```

Similarly for the include directory holding important named constants.

```
LOCAL_INCS := cbpmfio/include
```

In the previous version of this file access mechanism, several libraries and include directories were required. This is no longer the case. All references to BeamInstSupport and CBPM-TSHARC libraries, include directories, and or include statements with "cbi_" or "cbpm_" in the file name should be removed from the user's makefiles. References to GTK system libraries (typically via a pkg-config invocation in the makefile) may also be removed.

Using Matlab to read CBPM data files

Start Matlab using the default command syntax `matlab`.
A setup script is provided to actually load the libary needed into Matlab, but you must first make Matlab aware of this location so that the setup function can be found and run. This can be done with the following

```
  addpath([getenv('ACC_RELEASE_DIR') '/cbpmfio/matlab’]);
  load_CBPM_input();
```

API Definitions

For a given language, the API is divided into two parts.

1) The data reading mechanism: Reads data from disk and stores it in data structures for later
   reference.
2) Field access functions either return:
   a) copies of data values directly
      - or -
   b) references (pointers) to the start of pre-allocated multi-value data buffers

Mechanism Details

The data reading call performs the following specific actions:

- Searches for the requested file in the CESR ONLINE data directories, if not found there, it
- searches in the CESR OFFLINE data directories. If the file cannot be found, the reader function returns an error.
- Opens the file and reads in all supported fields and data buffers to memory
- Calculates all derived quantities from the raw data present in file
  - Gain-scaled and pedestal subtracted buffers are generated at this time.
- Closes file, leaving the data from a **single** file in memory, which can then be accessed by field/buffer using the provided access functions.
- If the reader function is called again, all data made available via the access functions is de-allocated and new data structures are spawned and populated with the data from the file specified to the most recent reader function call.

The list of fields available in the data file does not match 1-to-1 the list of field access functions.
Please see Appendix A for a list of all fields that appear in the supported data file format(s). If you require access in your software to fields that are supported, but for which no filed access function exists, please request an expansion of the API from the BPM instrumentation group.

Gray function prototypes are not yet implemented, but support is pending.

Note: The earliest file index that is supported by this input mechanism and the API functions defined below is RD-004080 Files earlier than this are not supported for read operations.

Postion Calculation

4-button Position Calculation Formula

Button labels

- TI: Top Inner
- BI: Bottom Inner
- TO: Top Outer
- BO: Bottom Outer

Xcoeff = Horizontal button spacing distance   (for CESR beam pipe: 0.0288m)
Ycoeff = Vertical button spacing distance   (for CESR beam pipe: 0.0223m)

sum   = TI + BI + BO + TO
Xnum = TO + BO - BI - TI
Ynum = TO - BO - BI + TI

X_position = Xcoeff * (Xnum/sum)   (in meters)
Y_position = Ycoeff * (Ynum/sum)   (in meters)

```
                                     ^
      TI=Card 0    TO=Card 3    | +Y
 IN
 <-
      BI=Card 1    BO=Card 2    +X
                                -->
```

C Language API

In order to provide function prototypes and access to certain important named constants for file reading operation function parameters, you must use the following include statement.

#include "cbpmfio.h"

This header reference makes the constants `CBPMFIO_MAX_STRING_LENGTH`, `CBPM_HORZ_DIM`, and `CBPM_VERT_DIM` available.

Note: The underscore at the end of two of these functions' names is required.

```
int   cbpm_read_rawfile(int *file_idx)
int   cbpm_read_rawfile_by_name_(char *full_filename, int len)

int   cbpm_rawfile_timestamp_(int *timestamp)
int   cbpm_rawfile_num_instruments(void)
int   cbpm_rawfile_num_bunches(void)
int   cbpm_rawfile_num_turns(void)
int   cbpm_rawfile_condx_number(void)
float cbpm_rawfile_cern_current(void)
int   cbpm_rawfile_inst_names_(char *name)
int   cbpm_rawfile_rf_bucket(int *bunch)
int   cbpm_rawfile_cesr_index_(char *inst_name)
float cbpm_rawfile_pedestal(char *inst_name, int *bunch, int *button)
float cbpm_rawfile_gain_factor(char *inst_name, int *bunch, int *button)
int   *cbpm_rawfile_raw_ptr(char *inst_name, int *button)
float *cbpm_rawfile_tbt_ptr(char *inst_name, int *button)
float *cbpm_rawfile_pos_ptr(char *inst_name, int *plane)
int   *cbpm_rawfile_phase_ptr(char *inst_name, int *plane)
```

```
//
// C test program for FIO mechanism
//
#include "cbpmfio.h"

int main(void) {

  int file_idx;
  int retval;
  int num_insts, num_bunches, num_turns;
  char timestamp[50] = "";
  float ff;
  int zero  = 0;
  int one   = 1;
  int two   = 2;
  int three = 3;
  int iidx, bunch, button;
  int *dat0, *dat1, *dat2, *dat3;
  float *fdat0, *fdat1, *fdat2, *fdat3;
  int rf_bucket;
  char name[20] = ".";
  int namecount = 0;
  char names[110][20];
```

```
//file_idx = 4080;
file_idx = 8360;
printf("F I R S T    F I L E\n");

retval = cbpm_read_rawfile(&file_idx);
printf("cbpm_read_rawfile_ returned: %d\n", retval);

num_insts = cbpm_rawfile_num_instruments();
num_bunches = cbpm_rawfile_num_bunches();
num_turns = cbpm_rawfile_num_turns();

cbpm_rawfile_timestamp_(timestamp);
printf("File ID     : %d\n", file_idx);
printf("TIMESTAMP   = %s\n", timestamp);
printf("NUM_INSTS   = %d\n", num_insts);
printf("NUM_BUNCHES = %d\n", num_bunches);
printf("NUM_TURNS   = %d\n", num_turns);
printf("CONDX #     = %d\n", cbpm_rawfile_condx_number());
ff = cbpm_rawfile_cern_current(); //necessary intermediate
printf("CERN CURR   = %f\n", ff);

cbpm_rawfile_inst_names_(name);
strcpy(names[namecount], name);
namecount++;
while(strcmp(name, "") != 0) {
  cbpm_rawfile_inst_names_(name);
  strcpy(names[namecount], name);
  namecount++;
}

for (iidx = 0; iidx < num_insts; iidx++) {
  printf("\n\nLocation: %s\n", names[iidx]);
  bunch = 1;
  rf_bucket = cbpm_rawfile_rf_bucket(&bunch);
  printf("RF bucket = %d\n", rf_bucket);
  printf("PEDS            %f %f %f %f\n",
         cbpm_rawfile_pedestal(names[iidx], &bunch, &zero),
         cbpm_rawfile_pedestal(names[iidx], &bunch, &one),
         cbpm_rawfile_pedestal(names[iidx], &bunch, &two),
         cbpm_rawfile_pedestal(names[iidx], &bunch, &three));

  printf("GAINS           %f %f %f %f\n",
         cbpm_rawfile_gain_factor(names[iidx], &bunch, &zero),
         cbpm_rawfile_gain_factor(names[iidx], &bunch, &one),
         cbpm_rawfile_gain_factor(names[iidx], &bunch, &two),
         cbpm_rawfile_gain_factor(names[iidx], &bunch, &three));

  dat0 = cbpm_rawfile_raw_ptr(names[iidx], &zero);
  dat1 = cbpm_rawfile_raw_ptr(names[iidx], &one);
  dat2 = cbpm_rawfile_raw_ptr(names[iidx], &two);
  dat3 = cbpm_rawfile_raw_ptr(names[iidx], &three);
  printf("RAW........... %d %d %d %d\n",
         *dat0,
          *dat1,
          *dat2,
          *dat3);
   printf("                %d %d %d %d\n",
          *(dat0+1),
          *(dat1+1),
          *(dat2+1),
          *(dat3+1));

   fdat0 = cbpm_rawfile_tbt_ptr(names[iidx], &zero);
   fdat1 = cbpm_rawfile_tbt_ptr(names[iidx], &one);
   fdat2 = cbpm_rawfile_tbt_ptr(names[iidx], &two);
   fdat3 = cbpm_rawfile_tbt_ptr(names[iidx], &three);
   printf("TBT............ %f %f %f %f\n",
          *fdat0,
          *fdat1,
          *fdat2,
          *fdat3);
   printf("                %f %f %f %f\n",
          *(fdat0+1),
          *(fdat1+1),
          *(fdat2+1),
          *(fdat3+1));


   fdat0 = cbpm_rawfile_pos_ptr(names[iidx], &zero);
   fdat1 = cbpm_rawfile_pos_ptr(names[iidx], &one);
   printf("POS x y         %f %f\n",
          *fdat0,
          *fdat1);
   printf("POS x y         %f %f\n",
          *(fdat0+1),
          *(fdat1+1));

   dat0 = cbpm_rawfile_phase_ptr(names[iidx], &zero);
   dat1 = cbpm_rawfile_phase_ptr(names[iidx], &one);
   printf("phase word x y  %3d %3d\n",
          *(dat0),
          *(dat1));
   printf("phase word x y  %3d %3d\n",
          *(dat0+1),
          *(dat1+1));


  } //endFOR iidx
  namecount = 0;

  return CBPMFIO_SUCCESS;
}
```

Fortran Language API

The following statements must appear at the beginning of any code that uses this API in order to make the functions described below available.

```
    use cbpm_file_input
```

In order to provide access to certain important named constants for file reading operation function parameters, you must use the following include statement.

```
    #include "cbpmfio_f_constants.h"
```

This header reference makes the constants `CBPMFIO_MAX_STRING_LENGTH`, `CBPM_HORZ_DIM`, `CBPM_VERT_DIM`, `CBPMFIO_SUCCESS`, and `CBPMFIO_FAILURE` available.

Functions:

```
Integer cbpm_read_rawfile_f(file_idx)
Integer cbpm_read_rawfile_by_name_f(filename)

Function cbpm_rawfile_timestamp_f(timestamp) result (stat)
Function cbpm_rawfile_num_instruments_f() result (val)
Function cbpm_rawfile_num_bunches_f() result (val)
Function cbpm_rawfile_num_turns_f() result (val)
Function cbpm_rawfile_condx_number_f() result (val)
Function cbpm_rawfile_cern_current_f() result (val)
Function cbpm_rawfile_inst_names_f(name) result (val)
Function cbpm_rawfile_rf_bucket_f(name) result (val)
Function cbpm_rawfile_cesr_index_f(name) result (retval)
Function cbpm_rawfile_pedestal_f(inst_name, bunch, button) result (val)
Function cbpm_rawfile_gain_factor_f(inst_name, bunch, button) result (val)
```

These functions require a pointer to be directed at a buffer location returned by the call.

```
Function cbpm_rawfile_raw_ptr_f(inst_name, button, dat_ptr) result (dptr)
Function cbpm_rawfile_tbt_ptr_f(inst_name, button, dat_ptr) result (dptr)
Function cbpm_rawfile_pos_ptr_f(inst_name, plane, dat_ptr) result (dptr)
Function cbpm_rawfile_phase_ptr_f(inst_name, plane, dat_ptr) result (dptr)
```

Fortran example program:
```
!
! Test program for calling a C function wrapped in a Fortran function for
! reading a raw data file (RD-V3 or later).
!
#include "cbpmfio_f_constants.h"

PROGRAM cbpmfio_test_f

  use cbpm_file_input

  type (c_ptr) c_pointer

  integer, pointer :: raw_array0(:)
  integer, pointer :: raw_array1(:)
  integer, pointer :: raw_array2(:)
  integer, pointer :: raw_array3(:)

  Integer retval, num_bunches, num_turns, bunch, inst, val, condx_number
  Real fretval0, fretval1, fretval2, fretval3
  Real*4 cern_curr

  Character*(40) :: timestamp
  Character*(6)  :: inst_name

  Real*4, pointer :: tbt_array0(:)
  Real*4, pointer :: tbt_array1(:)
  Real*4, pointer :: tbt_array2(:)
  Real*4, pointer :: tbt_array3(:)

  Real*4, pointer :: pos_array_h(:)
  Real*4, pointer :: pos_array_v(:)

  Integer, pointer ::  phase_array_h(:)
  Integer, pointer ::  phase_array_v(:)

  Integer num_instruments
  Integer zero
  Integer one
  Integer two
  Integer three
  Integer file_idx
  Integer num_words
  Integer rf_bucket

  bunch = 1
  zero = 0
  one = 1
  two = 2
  three = 3

  ! Read data from file, specified by file index number.
  print *,"F I R S T    F I L E"
  file_idx = 4080
```

```
retval = cbpm_read_rawfile_f( file_idx )
print *,"cbpm_read_rawfile returned: ",retval

! Get key numerical values
retval = cbpm_rawfile_timestamp_f(timestamp)

num_instruments = cbpm_rawfile_num_instruments_f()
num_bunches = cbpm_rawfile_num_bunches_f()
num_turns = cbpm_rawfile_num_turns_f()
condx_number = cbpm_rawfile_condx_number_f()
cern_curr = cbpm_rawfile_cern_current_f()

print *,"File ID     : ", file_idx
print *,"TIMESTAMP   = ", timestamp
print *,"NUM_INSTS   = ", num_instruments
print *,"NUM BUNCHES = ", num_bunches
print *,"NUM TURNS   = ", num_turns
print *,"CONDX #     = ", condx_number
print *,"CERN CURR   = ", cern_curr
print *,""
print *,""

num_words = num_bunches * num_turns
inst_name = "."

do while (inst_name(1:1) .NE. " ")

   retval = cbpm_rawfile_inst_names_f(inst_name)
   if (inst_name .EQ. " ") exit
   print *,"Location: ",inst_name

   rf_bucket = cbpm_rawfile_rf_bucket_f(three)
   print *,"RF bucket      ", rf_bucket

   fretval0 = cbpm_rawfile_pedestal_f(inst_name, bunch, zero)
   fretval1 = cbpm_rawfile_pedestal_f(inst_name, bunch, one)
   fretval2 = cbpm_rawfile_pedestal_f(inst_name, bunch, two)
   fretval3 = cbpm_rawfile_pedestal_f(inst_name, bunch, three)
   print *,"PEDS            ", fretval0, fretval1, fretval2, fretval3

   fretval0 = cbpm_rawfile_gain_factor_f(inst_name, bunch, zero)
   fretval1 = cbpm_rawfile_gain_factor_f(inst_name, bunch, one)
   fretval2 = cbpm_rawfile_gain_factor_f(inst_name, bunch, two)
   fretval3 = cbpm_rawfile_gain_factor_f(inst_name, bunch, three)
   print *,"GAINS           ", fretval0, fretval1, fretval2, fretval3

   raw_array0 => cbpm_rawfile_raw_ptr_f(inst_name, zero)
   raw_array1 => cbpm_rawfile_raw_ptr_f(inst_name, one)
   raw_array2 => cbpm_rawfile_raw_ptr_f(inst_name, two)
   raw_array3 => cbpm_rawfile_raw_ptr_f(inst_name, three)
   print *, "RAW........... ", raw_array0(1), raw_array1(1),
                               raw_array2(1), raw_array3(1)
   print *, "                ", raw_array0(2), raw_array1(2),
                               raw_array2(2), raw_array3(2)
```

```
     tbt_array0 => cbpm_rawfile_tbt_ptr_f(inst_name, zero)
     tbt_array1 => cbpm_rawfile_tbt_ptr_f(inst_name, one)
     tbt_array2 => cbpm_rawfile_tbt_ptr_f(inst_name, two)
     tbt_array3 => cbpm_rawfile_tbt_ptr_f(inst_name, three)
     print *,"TBT............ ", tbt_array0(1), tbt_array1(1),
                                tbt_array2(1), tbt_array3(1)
     print *,"TBT             ", tbt_array0(2), tbt_array1(2),
                                tbt_array2(2), tbt_array3(2)

     pos_array_h => cbpm_rawfile_pos_ptr_f(inst_name, CBPM_HORZ_DIM)
     pos_array_v => cbpm_rawfile_pos_ptr_f(inst_name, CBPM_VERT_DIM)
     print *,"POS x y         ", pos_array_h(1), pos_array_v(1)
     print *,"                ", pos_array_h(2), pos_array_v(2)

     phase_array_h => cbpm_rawfile_phase_ptr_f(inst_name, CBPM_HORZ_DIM)
     phase_array_v => cbpm_rawfile_phase_ptr_f(inst_name, CBPM_VERT_DIM)
     print *,"phase word x y  ", phase_array_h(1), phase_array_v(1)
     print *,"phase word x y  ", phase_array_h(2), phase_array_v(2)

     print *,""
     print *,""

  end do
end program
```

Matlab Language API

In order to make these functions available from within Matlab, you must invoke the following
early in your session or .m file.

```
  addpath([getenv('ACC_RELEASE_DIR') '/cbpmfio/matlab']);
  load_CBPM_input();
```

```
double cbpm_read_rawfile_m(file_index)
double cbpm_read_rawfile_by_name_m(char *full_filename, int len)

double cbpm_file_timestamp_m()
double cbpm_file_num_insts_m()
double cbpm_file_num_bunches_m()
double cbpm_file_num_turns_m()
double cbpm_file_condx_m()
double cbpm_file_current_m()
string cbpm_file_inst_locs_m()
double cbpm_rawfile_rf_bucket_(bunch)
double cbpm_rawfile_cesr_index_(location)
double cbpm_file_pedestal_m(location, bunch, button)
double cbpm_file_gain_factor_m(location, bunch, button)
matrix cbpm_file_raw_data_m(location)
matrix cbpm_file_tbt_data_m(location)
matrix cbpm_file_pos_data_m(location)
matrix cbpm_file_shaker_phase_m(location)
```

```
function [] =  cbpmFileInputDemoExtended( )

    addpath([getenv('ACC_RELEASE_DIR') '/cbpmfio/matlab']); load_CBPM_input();

    TEST_FILE_IDX  =   4080;

    file_idx = TEST_FILE_IDX;

  fprintf('\nAttempting to open RAW DATA file %d from central CESR datafile
location...\n', file_idx);

  status = cbpm_read_rawfile_m( file_idx );
  if (status ~= 0)
    fprintf( 'Error opening file for reading... exiting.\n');
    return
  end

  button  = 0;
  button0 = 0;
  button1 = 1;
  button2 = 2;
  button3 = 3;

  fprintf('Testing cbpm_file_timestamp_:\n');
  timestamp2 = cbpm_file_timestamp_m(  );
  fprintf('  timestamp = %s\n', timestamp2 );

  num_instruments = cbpm_file_num_insts_m();
  fprintf('Testing cbpm_file_num_instruments_:\n');
  fprintf('  num_instruments = %d\n', num_instruments );
  fprintf('Testing cbpm_file_num_bunches_:\n');
  fprintf('  num_bunches     = %d\n', cbpm_file_num_bunches_m() );

  fprintf('Testing cbpm_file_num_turns_:\n');
  fprintf('  num_turns       = %d\n', cbpm_file_num_turns_m() );

  fprintf('Testing cbpm_file_condx_number_:\n');
  fprintf('  CONDX number    = %d\n', cbpm_file_condx_m() );

  names{1} = '';
  namecount = 1;
  name = cbpm_file_inst_locs_m(  );
  while ( strcmp(name, '') == 0 )
    names{namecount} = name ;
    namecount = namecount + 1;
    name = cbpm_file_inst_locs_m(  )
  end

  for count = 1:num_instruments

    location = names{count} ;
    fprintf('\n\nLocation = %s\n', location);

    button0 = 0;
    bunch = 0;
    fprintf('PED: %f   %f   %f   %f\n', ...
           cbpm_file_pedestal_m(location, bunch, button0), ...
           cbpm_file_pedestal_m(location, bunch, button1), ...
           cbpm_file_pedestal_m(location, bunch, button2), ...
           cbpm_file_pedestal_m(location, bunch, button3) ...
           );
   fprintf('Gain factor: %f   %f   %f   %f\n', ...
           cbpm_file_gain_factor_m(location, bunch, button0), ...
           cbpm_file_gain_factor_m(location, bunch, button1), ...
           cbpm_file_gain_factor_m(location, bunch, button2), ...
           cbpm_file_gain_factor_m(location, bunch, button3) ...
           );

    idata = cbpm_file_raw_data_m( location  );
    fprintf('raw : %d %d %d %d \n', idata);

    fdata = cbpm_file_tbt_data_m( location);
    fprintf('TBT: %f %f %f %f \n', fdata );

    pos = cbpm_file_pos_data_m( location);%, dim, fdata );
    fprintf('X pos:  %f, Y pos:  %f\n', pos );

    phase = cbpm_file_shaker_phase_m( location);%, dim, idata );
    fprintf('Phase word: %x %x\n', phase );

  end

  file_idx = 4079;
  fprintf('\nAttempting to open RAW DATA file %d from central CESR datafile
location...\n', file_idx);

  status = cbpm_read_rawfile_m( file_idx );
  if (status ~= 0)
```

```
    printf( 'Error opening file for reading... exiting.\n');
    return;
  end

  fprintf('Demo complete.');

end
```

API Description

The following are detailed descriptions of each API function with language-specific implementation details where necessary.

Read Functions

These take a file index value and attempt to retrieve the file with that index from the CESR ONLINE data storage area. Failing that, it will try the CESR OFFLINE data storage area before throwing an error.

```
Int cbpm_read_rawfile(int *file_idx)
Integer cbpm_read_rawfile_f(file_idx)
cbpm_read_rawfile_m(file_index)
```

Argument:
- Integer representing the RAW data file index to load

Return value:
CBPMFIO_SUCCESS (0) - If function completed successfully
CBPMFIO_FAILURE (1) - If function experienced an error opening the file

Note:
The internal data structures that hold raw data, and the structures that contain automatically calculated values based on this raw data (scaled button values, X/Y positions, etc...) are populated upon every call to cbpm_read_rawfile. The data will need to be copied to another location provided by the developer if one wishes to preserve a particular set of data when a new call to cbpm_read_rawfile is made.

Accessor Functions

Once the data has been imported and supplementary data buffers automatically calculated, one may access any of these values by using the appropriate accessor function.

```
int cbpm_rawfile_timestamp(char *timestamp)
Function cbpm_rawfile_timestamp_f( timestamp ) result (retval)
function string = cbpm_raw_timestamp_m( )
```

Returns a timestamp string (by argument reference or as return value, depending on language) containing the time the measurement was performed. The string returned has a maximum length of `CBI_MAX_STRING_LENGTH`.

```
int      cbpm_rawfile_num_instruments_( void )
Function cbpm_rawfile_num_instruments_f( ) result (retval)
function scalar = cbpm_file_num_instruments_m( )
```

Returns the number of instruments that provided data for the measurement.

```
int      cbpm_rawfile_num_bunches_( void )
Function cbpm_rawfile_num_bunches_f( ) result (retval)
function scalar = cbpm_file_num_bunches_m( )
```

Returns the number of bunches acquired in the measurement data file.

```
int      cbpm_rawfile_num_turns_( void )
Function cbpm_rawfile_num_turns_f( ) result (retval)
function scalar = cbpm_file_num_turns_m( )
```

Returns the number of turns acquired for each bunch in the imported measurement data.

```
int      cbpm_rawfile_condx_number_( void )
Function cbpm_rawfile_condx_number_f( ) result (retval)
function scalar = cbpm_file_condx_m( )
```

Returns the CESR conditions (CONDX) file index value in place when the measurement was taken.

The CONDX number is the value obtained from the node "RELATED SETS" element 12 in the CESR database.

```
float    cbpm_rawfile_cern_current_( void )
Function cbpm_rawfile_cern_current_f( ) result (retval)
function scalar = cbpm_file_current_m( )
```

Returns a floating point value representing the CESR CERN total current measurement.

```
int      cbpm_rawfile_inst_names_( char *name )
Function cbpm_rawfile_inst_names_f( name ) result (retval)
function  cbpm_file_inst_locs_m( )
```

Upon each call, returns a single string of the next instrument found in the data file. "name" will be NULL (blank string for Fortran) for the call made immediately after the last ID string pointer has been returned. At this point, the caller will have seen every instrument name present in the data file. There is no need to make another call until cbpm_read_rawfile is called for a different data file. If this function is called again after returning a NULL or blank string, it will simply start the instrument name listing process again returning names starting from the beginning of the file and iterating through them.

Name strings are of the form "12W", "12W2", "8AW", "21E", etc...

Arguments:
- Pointer to string able to receive an instrument name by action of this function.

```
int      cbpm_rawfile_rf_bucket_( int *bunch )
Function cbpm_rawfile_rf_bucket_f( bunch) result (retval)
function  cbpm_file_rf_bucket_m( bunch )
```

Returns the CESR RF bucket number for the bunch index provided. The bunch indexing is dependent upon the timing setup that the instrument was in when the measurement was taken. This function takes that fact into account automatically.

```
float cbpm_rawfile_pedestal_(char *inst_location, int *bunch, int *button)
Function cbpm_rawfile_pedestal_( inst_location, bunch, button ) result (retval)
function cbpm_file_pedestal_m(location, bunch, button)
```

Returns the pedestal value stored for the given bunch index and button.

```
float cbpm_rawfile_gain_factor_( char *inst_location, int *bunch, int *button )
Function cbpm_rawfile_gain_factor_( inst_location, bunch, button ) result (retval)
function cbpm_file_gain_factor_m(location, bunch, button)
```

Returns the gain scale factor stored for the given bunch index and button.

Arguments:
- Pointer to the instrument ID / location string for the BPM desired
- Bunch index (First bunch acquired is index 1)
  - Button index (0,1,2,3)
  -

```
int      cbpm_rawfile_raw_ptr_( char *inst_location, int *button, int *dat_ptr )
Function cbpm_rawfile_raw_ptr_f( inst_location, button, dat_ptr ) result (retval)
function cbpm_file_raw_data_m( location )
```

Returns an integer pointer (array) to the first buffer location of a particular BPM's signal input buffer (i.e. button). These values are the RAW ADC values acquired.

Data organization:
The data are ordered in the following fashion . For each bunch that is digitized, the data for all turns appears in order, then so on for each subsequent bunch. This scheme applies to raw data obtained via this function as well as turn-by-turn (TBT, ped-subtracted and scaled) data, and position data accessed by the next several functions described below.

Arguments:
- Pointer to the instrument ID / location string for the BPM desired
- Pointer to integer value for the button (card) from where the data came.
  Valid values are <0, 1, 2, 3>
- Pointer to (int) data array to fill

```
int      cbpm_rawfile_tbt_ptr_( char *inst_location, int *button, int *dat_ptr )
Function cbpm_rawfile_tbt_ptr_f( inst_location, button, dat_ptr ) result (retval)
function cbpm_file_tbt_data_m( location )
```

Returns a float pointer (array) to the first buffer location of a particular BPM's signal input buffer (i.e. button). These values are PEDESTAL-SUBTRACTED and GAIN-SCALED already, i.e. in "physics units".

Data organization:
The data are ordered in the following fashion
For each bunch that is digitized, the data for all turns appears in order, then
so on for each subsequent bunch.

Arguments:
- Pointer to the instrument ID / location string for the BPM desired
- Pointer to integer value for the button (card) from where the data came.
  Valid values are <0, 1, 2, 3>
- Pointer to (float) data array to fill

```
int      cbpm_rawfile_pos_ptr_( char *inst_location, int *plane, int *dat_ptr )
Function cbpm_rawfile_pos_ptr_f( inst_location, plane, dat_ptr ) result (retval)
function cbpm_file_pos_data_m( location )
```

Returns a float pointer (array) to the first buffer location of a particular BPM's position value for the plane specified, either horizontal or vertical. The positions are not stored in the data file, but are calculated and stored in memory upon execution of the Part 1 import function. The positions are calculated for every turn acquired using the fully corrected PEDESTAL-SUBTRACTED and GAIN-SCALED values for the gain setting employed during acquisition.

Data organization:
The data are ordered in the following fashion
For each bunch that is digitized, the data for all turns appears in order, then
so on for each subsequent bunch.

Arguments:
- Pointer to the instrument ID / location string for the BPM desired
- Integer value for the transverse plane of beam displacement to access
  Valid values are <CBPM_HORZ_DIM, CBPM_VERT_DIM>
- Pointer to (float) data array to fill

```
int      cbpm_rawfile_phase_ptr_( char *inst_location, int *plane, int *dat_ptr )
Function cbpm_rawfile_phase_ptr_f( inst_location, plane, dat_ptr ) result (retval)
function cbpm_file_shaker_phase_m( location )
```

Returns an integer pointer (Array) to the first buffer location (first turn) of a particular BPM's encoded shaker phase word. This is the bit pattern encoded onto the command word embedded into the CESR clock signal turns marker burst. The bit pattern is made available in two parts, one for the encoded horizontal shaker phase and the other for vertical.

Data organization:
  The data are ordered in the following fashion
  For each bunch that is digitized, the data for all turns appears in order, then
  so on for each subsequent bunch.

Arguments:
- Pointer to the instrument ID / location string for the BPM desired
- Integer value for the transverse plane of shaker phase to access
  Valid values are <CBPM_HORZ_DIM, CBPM_VERT_DIM>
- Pointer to (float) data array to fill

Appendix A – Data File Header Fields

```
Instrument_Type
Command_ID
File_type
File_ID
File_Version
Timestamp
Core_commstruct_v
Plat_commstruct_v
Bunch_Patt_Name
Bunch_Patt_(hex)
Species
Num_Instruments
Number_of_Bunches
Number_of_Turns
Turn_Spacing
Timing_Setup
Trigger
CESR CONDX
CERN Current Raw
CERN Current mA
Location
BPM_hostname
BPM_IP_Address
Detector_Type
Detector_Coeffs
EXE_Name
EXE_Build_ID
Dig.Board_FPGA
Front-End_FPGAs
Timing_Setup
Number_of_Turns
Turn_sync_counter
Turn_spacing
Trigger
Bunch_Pat_offsets
Com_Turnmrk_Dly
Blk_Turnmrk_Dlys
Block_Delays
Channel_Delays
Gain_Settings
Gain_Codes
Gain_Coeffs
Pedestals
Digital_Temp_C
Card_Temps_C
ADC_saturation
ADC_High
ADC_Low
```
Button Data Values...