CcmAccess

CcmApi Reference Manual

Version 2.0

Table of Contents

Table of Contents

System Requirements

The CcmAccess software has the following system requirements:

- Windows 7 or higher
- 64-bit processor
- At least one free USB port

Revision Notes

This manual, version 2.0, reflects the changes to the CcmApi from version 1.1x to current.

- Added support for more than 40 microphones
- Added camera calibration when requesting images through the API
- Supports hardware with different sample rate clocks (51.2kHz instead of 50.0kHz)
- Can request the number of frames available from source logfiles
- Can request images to be synchronized with acoustic data
- Changes to configuration XML to support camera settings
- Added many new functions for camera and image settings

New functions:

Camera control

```
CcmCaptureImage
CcmGetCameraEnabled
CcmGetCameraId
CcmGetFramesPerImage
CcmSetCameraEnabled
CcmSetCameraId
CcmSetFramesPerImage
```

Image settings

```
CcmGetCameraResolution
CcmGetImageBrightness
CcmGetImageColorMode
CcmGetImageExtents
CcmGetImageFormat
CcmGetImageSize
CcmGetJpgCompression
CcmGetPngCompression
CcmSetCameraResolution
CcmSetImageBrightness
CcmSetImageColorMode
CcmSetImageFormat
CcmSetJpgCompression
CcmSetPngCompression
```

Array Information

```
CcmGetArrayDescription
CcmGetArrayModel
CcmGetArraySerialNumber
```

These functions return the information for the assembled array. The functions CcmGetDescription, CcmGetModel and CcmGetSerialNumber return the information for the CCM controller module. These functions will continue to be supported, but are considered to be deprecated.

Other

CcmGetCurrentLogfileFolder
CcmGetSampleRateList
CcmGetTimeDomainFrameOrder
CcmRead
CcmSetTimeDomainFrameOrder

Removed functions:

CcmGetFrequencyDomainFrameBufferEnabled
CcmGetTimeDomainFrameBufferEnabled
CcmSetFrequencyDomainFrameBufferEnabled
CcmSetTimeDomainFrameBufferEnabled

The functionality of the removed "FrameBufferEnabled" functions can be implemented using the corresponding "FrameBufferSize" functions. Setting the frame buffer size to zero is equivalent to disabling the frame buffer, and setting the size to non-zero is equivalent to enabling the frame buffer.

Renamed functions:

CcmGetOverlapEnabled is replaced with CcmGetFrequencyDomainOverlapEnabled
CcmGetLogfilePath is replaced with CcmGetLogfileRoot
CcmIsDeviceAvailable is replaced with CcmGetAvailabilityStatus
CcmIsDeviceOpened is replaced with CcmGetDeviceHandleStatus
CcmIsUsbConnected is replaced with CcmGetUsbConnectionStatus
CcmSetOverlapEnabled is replaced with CcmSetFrequencyDomainOverlapEnabled
CcmSetLogfilePath is replaced with CcmSetLogfileRoot

Deprecated functions:

CcmGetDescription should be replaced with CcmGetArrayDescription for new applications
CcmGetModel should be replaced with CcmGetArrayModel for new applications
CcmGetSerialNumber should be replaced with CcmGetArraySerialNumber for new applications
CcmReadFrames should be replaced with CcmRead for new applications

Introduction to the API

This chapter defines and describes some of the features used with the API. The API functions are described in detail later in the manual.

Time Domain / Frequency Domain

The CCM features built-in real-time FFT output allowing for data in both the time and frequency domains.

Time domain data is the calibrated sampled data. It is generally more versatile in that it allows for reprocessing.

Frequency domain data is the result of processing the time domain data with a real-time FFT. It removes the need for an external FFT which makes it a better choice for real-time beamforming operations.

Both time-domain and frequency-domain data can be returned from a single read operation. This allows real-time beamforming and time-domain processing or logging at the same time.

Data Organization

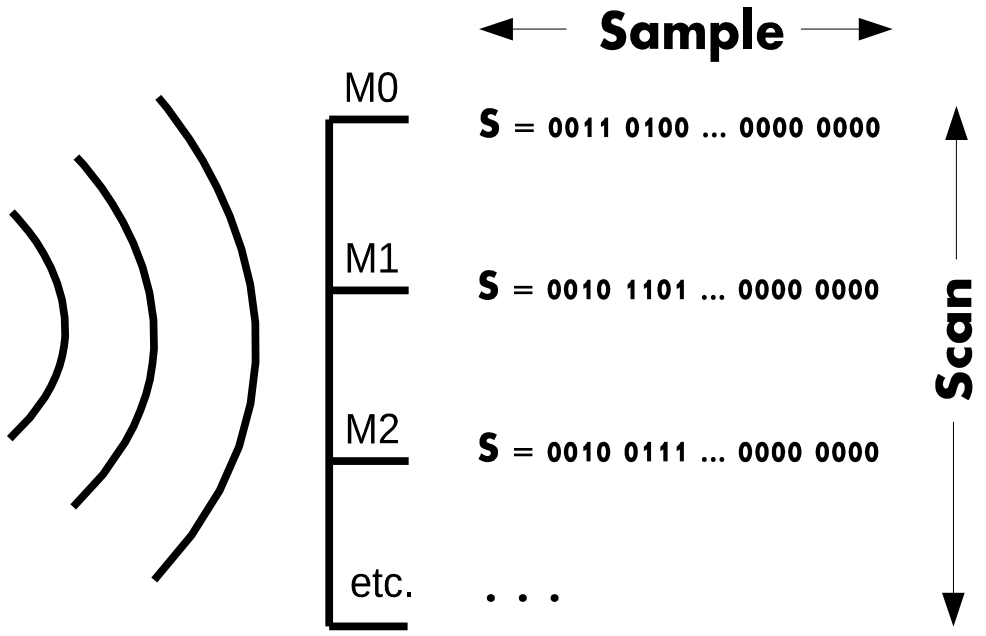
The CCM system organizes data into four catagories: samples, scans, blocks and frames.

Sample

A sample is a single digitized measurement from one microphone. Each sample is a 32-bit value with 24-bit resolution (the lower 8 bits are padded with zeros). The sample rate is controlled by the function `CcmSetSampleRate`.

Scan

A scan consists of one sample from each available microphone. The number of microphones can be found using the function `CcmGetMicrophoneCount`. Since the CCM samples all microphones simultaneously, the scan rate is equivalent to the sample rate.



Samples and Scans

Block

FFT operations require time-domain data to be grouped into some number of samples called blocks. The number of samples in a block, N, is a power of two. The resulting data is a set of N/2 complex values with real and imaginary components, in the frequency domain. The FFT block size is set using the function `CcmSetFrequencyDomainBlockSize`.

The time-domain data block size is set using the function `CcmSetTimeDomainBlockSize`. The block size determines the order in which the time domain data arrive.

Frame

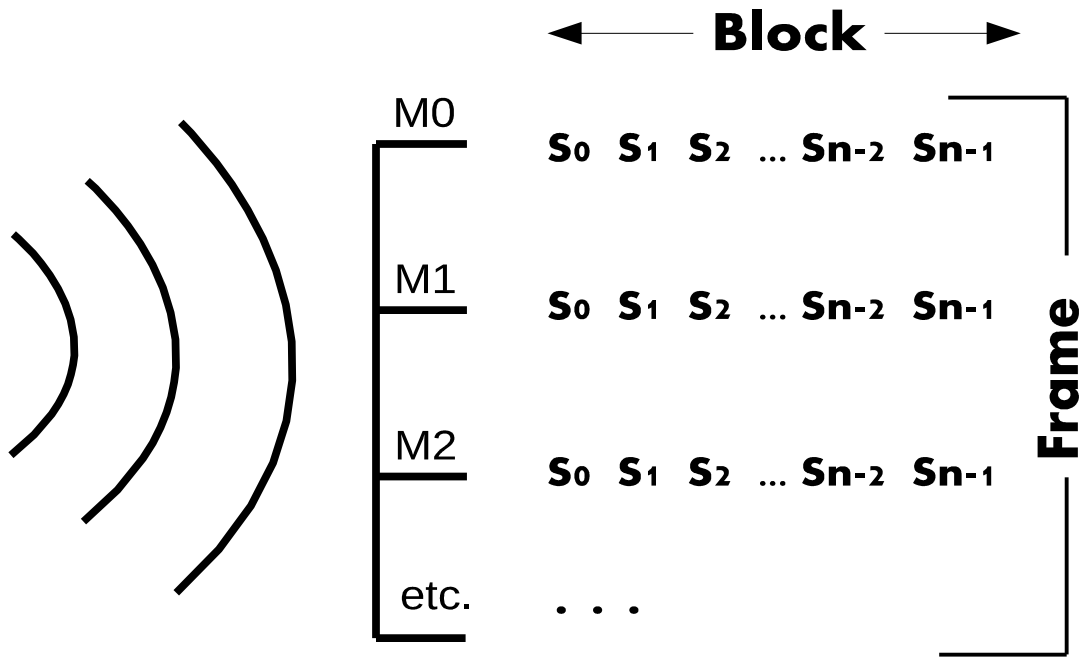
A Frame consists of N * M samples, where N = the block size and M = the number of microphones. It is the base unit required for any beamforming operation.

The frame has a slightly different meaning with time-domain and frequency-domain data.

A time-domain frame can be thought of as either:

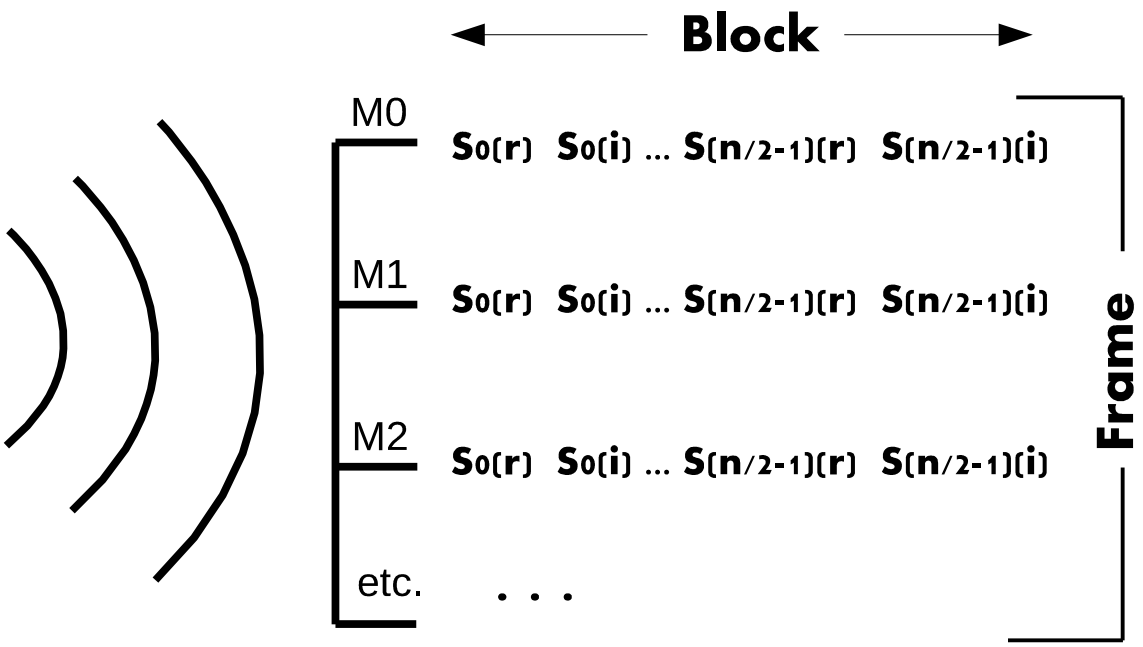
- One Block from each microphone; "Block Order"
- N Scans, where N is the block size; "Scan Order"

By default, time-domain data is returned in Block Order, but can be converted to Scan Order by calling the function `CcmSetTimeDomainFrameOrder(CCM_FRAMEORDER_SCAN)`.



Blocks and Frames (time-domain frame shown)

A frequency-domain frame is always one-half block of complex pairs of FFT data from each microphone. The second half of an FFT is always a mirror image of the first half. Removing the redundant second half allows the time-domain and frequency-domain frame sizes to be the same.



Blocks and Frames (frequency-domain frame shown)

Frame Data Order

In the table shown below, the data is in Block Order. The numbers in the table refer to the sequence in which the individual samples arrive. Note that one full block from each microphone is returned before any data from the next microphone.

Block Order is the default for the system. It simplifies applying the time-domain data to an external FFT function, but makes it difficult to graph a time-domain series. It is possible to convert Block Order to Scan Order by transposing each frame as it is read from the DLL using the function `CcmSetTimeDomainFrameOrder`.

	SAMPLE								
MICROPHONE		**s(0)**	**s(1)**	**s(2)**	**s(3)**	**...**	**s(N-3)**	**s(N-2)**	**s(N-1)**
	Frame 0								
	m_0	0	1	2	3	...	N-3	N-2	N-1
	m_1	N	N+1	N+2	N+3	...	2N-3	2N-2	2N-1
	m_2	2N	2N+1	2N+2	2N+3	...	3N-3	3N-2	3N-1
	...								
	m(M-1)	(M-1)N	(M-1)N+1	(M-1)N+2	(M-1)N+3	...	MN-3	MN-2	MN-1
	Frame 1								
	m_0	F+0	F+1	F+2	F+3	...	F+N-3	F+N-2	F+N-1
	m_1	F+N	F+N+1	F+N+2	F+N+3	...	F+2N-3	F+2N-2	F+2N-1
	m_2	F+2N	F+2N+1	F+2N+2	F+2N+3	...	F+3N-3	F+3N-2	F+3N-1
	...								
	m(M-1)	F+ (M-1)N	F+ (M-1)N+1	F+ (M-1)N+2	F+ (M-1)N+3	...	F+MN-3	F+MN-2	F+MN-1
	Frame 2								
	...	...	...	...	...	...	...	...	...

Block Order data

```
s = sample number
m = microphone number
N = time-domain block size
M = total number of microphones
F = frame size = N * M
```

In the table shown below, the data is in Scan Order. The numbers in the table refer to the sequence in which the individual samples arrive. Note that a full scan of one sample from each microphone is returned before any data from the next scan.

Scan Order data is obtained with the function
`CcmSetTimeDomainFrameOrder(CCM_FRAMEORDER_SCAN);`

		SAMPLE							
MICROPHONE		**s(0)**	**s(1)**	**s(2)**	**s(3)**	...	**s(N-3)**	**s(N-2)**	**s(N-1)**
	Frame 0								
	m_0	0	M	2M	3M	...	(N-3)M	(N-2)M	(N-1)M
	m_1	1	M+1	2M+1	3M+1	...	(N-3)M+1	(N-2)M+1	(N-1)M+1
	m_2	2	M+2	2M+2	3M+2	...	(N-3)M+2	(N-2)M+2	(N-1)M+2
	...								
	m(M-1)	M-1	2M-1	3M-1	4M-1	...	(N-2)M-1	(N-1)M-1	NM-1
	Frame 1								
	m_0	F+0	F+M	F+2M	F+3M	...	F+ (N-3)M	F+ (N-2)M	F+ (N-1)M
	m_1	F+1	F+M+1	F+2M+1	F+3M+1	...	F+ (N-3)M+1	F+ (N-2)M+1	F+ (N-1)M+1
	m_2	F+2	F+M+2	F+2M+2	F+3M+2	...	F+ (N-3)M+2	F+ (N-2)M+2	F+ (N-1)M+2
	...								
	m(M-1)	F+M-1	F+2M-1	F+3M-1	F+4M-1	...	F+ (N-2)M-1	F+ (N-1)M-1	2F-1
	Frame 2								
	...	...	...	...	...	...	...	...	...

Scan Order

```
s = sample number
m = microphone number
N = time-domain block size
M = total number of microphones
F = frame size = N * M
```

Frequency-domain data are returned as complex values with real and imaginary components as shown in the table below.

There is no option to transpose frequency-domain data.

Note that, because of the symmetry of the FFT results, only the first half of the data is returned. This allows the frequency-domain and time-domain frames to be the same size.

		SAMPLE						
MICROPHONE		**$X_{(0)}re$**	**$X_{(0)}im$**	**$X_{(1)}re$**	**$X_{(1)}im$**	...	**$X_{(N/2-1)}re$**	**$X_{(N/2-1)}im$**
	Frame 0							
	m_0	0	1	2	3	...	N-2	N-1
	m_1	N	N+1	N+2	N+3	...	2N-2	2N-1
	m_2	2N	2N+1	2N+2	2N+3	...	3N-2	3N-1
	...							
	$m_{(M-1)}$	(M-1)N	(M-1)N+1	(M-1)N+2	(M-1)N+3	...	MN-2	MN-1
	Frame 1							
	m_0	F+0	F+1	F+2	F+3	...	F+N-2	F+N-1
	m_1	F+N	F+N+1	F+N+2	F+N+3	...	F+2N-2	F+2N-1
	m_2	F+2N	F+2N+1	F+2N+2	F+2N+3	...	F+3N-2	F+3N-1
	...							
	$m_{(M-1)}$	F+ (M-1)N	F+ (M-1)N+1	F+ (M-1)N+2	F+ (M-1)N+3	...	F+MN-2	F+MN-1
	Frame 2							
	...	...	...	...	...	...	...	...

Frequency-Domain Data Frame

```
X = frequency-domain complex value
m = microphone number
N = time-domain block size
M = total number of microphones
F = frame size = N * M

re = real component of complex value
im = imaginary component of complex value
```

Features

In addition to time-domain sampling and real-time FFT data, the CcmAccess software provides additional features that allow an application to spend more time on processing instead of data management.

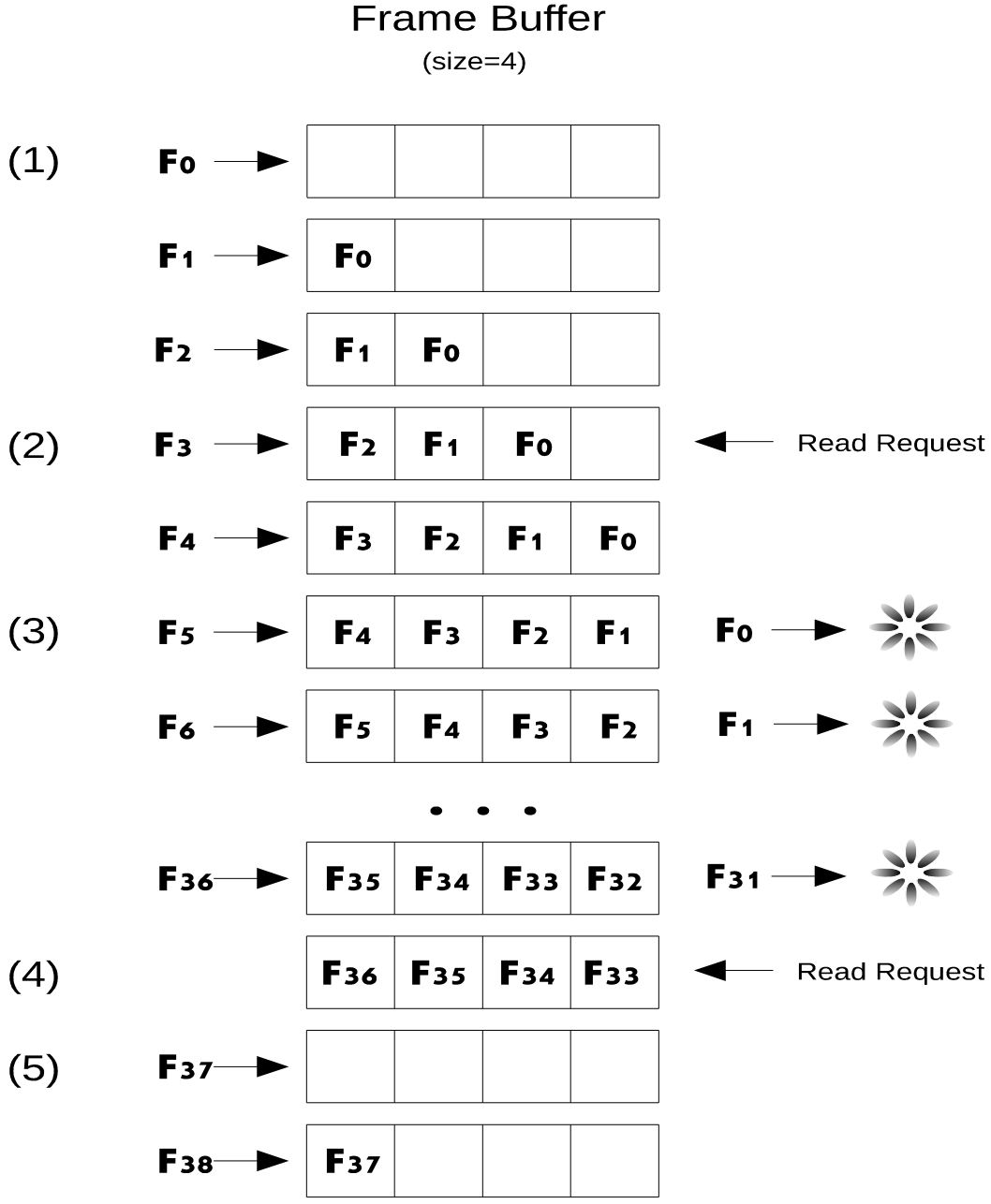
Frame Buffers

A frame buffer is a circular buffer of frames. The size of the frame buffer, or the number of frames it contains, is configured using the API functions `CcmSetTimeDomainFrameBufferSize` and `CcmSetFrequencyDomainFrameBufferSize`. Only the most recent frames are stored; older frames are discarded. There are separate frame buffers for time domain and frequency domain data.

Frame buffers are used when multiple frames which are closely related in time are necessary to perform a calculation. Complex calculations using several frames may be slow and cause the incoming sampled data to back up, or possibly overflow. Frame buffers allow these calculations to always use the most recent data while automatically discarding data that would otherwise make the application unresponsive.

When frame buffers are enabled, data will be returned to the application only when a full buffer is available. The API function `CcmRead`, the same function used to read single frames, is used to read frame buffers. If frame buffers are enabled, the application must supply a data buffer large enough to hold the full frame buffer. After the frame buffer has been read, it is cleared and begins acquiring new data for the next request.

The following figure describes frame buffer operation.

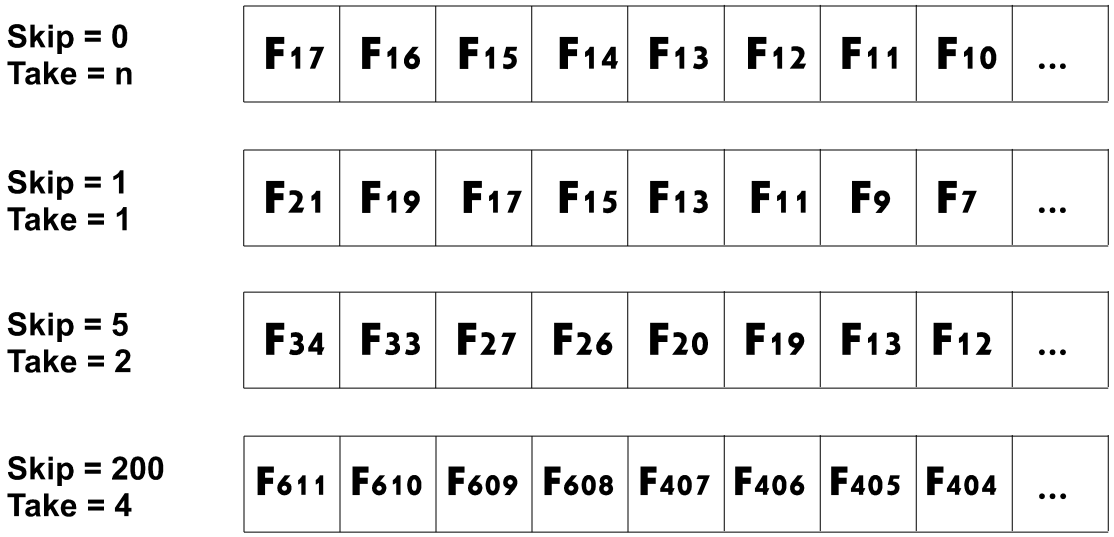


1. At startup, the frame buffer is empty.
2. The application tries to read the frame buffer before it has a full set of frames. Zero frames are returned.
3. When the frame buffer is full, the oldest frames are discarded to make room for new frames
4. The application reads a full frame buffer. All frames in the buffer are returned.
5. After a successful read, the frame buffer is cleared and new frames begin to accumulate.

Skip and Take

Skip and Take functions reduce the overall data rate if an application cannot keep up with the full data rate, and missing some frames is allowable. This is similar to using Frame Buffers, but Frame Buffers return data only when requested by the application while Skip and Take return data continuously which must be handled by the application.

When using Skip and Take, the application should always read frames in multiples of the value of Take. In the last example below, reading three frames at a time instead of four would result in some of the data being separated in time by 200 frames.



Skip and Take vs Frame Buffers

Skip and Take

Advantages

- The data rate is consistent so the time interval between the frames can be calculated exactly
- Done at the hardware level reducing amount of data sent over the USB

Disadvantages

- Data must be processed by the application to prevent overflow
- Increases application workload

Frame Buffers

Advantages

- Data does not need to be processed until the application is ready for it
- Decreases application workload

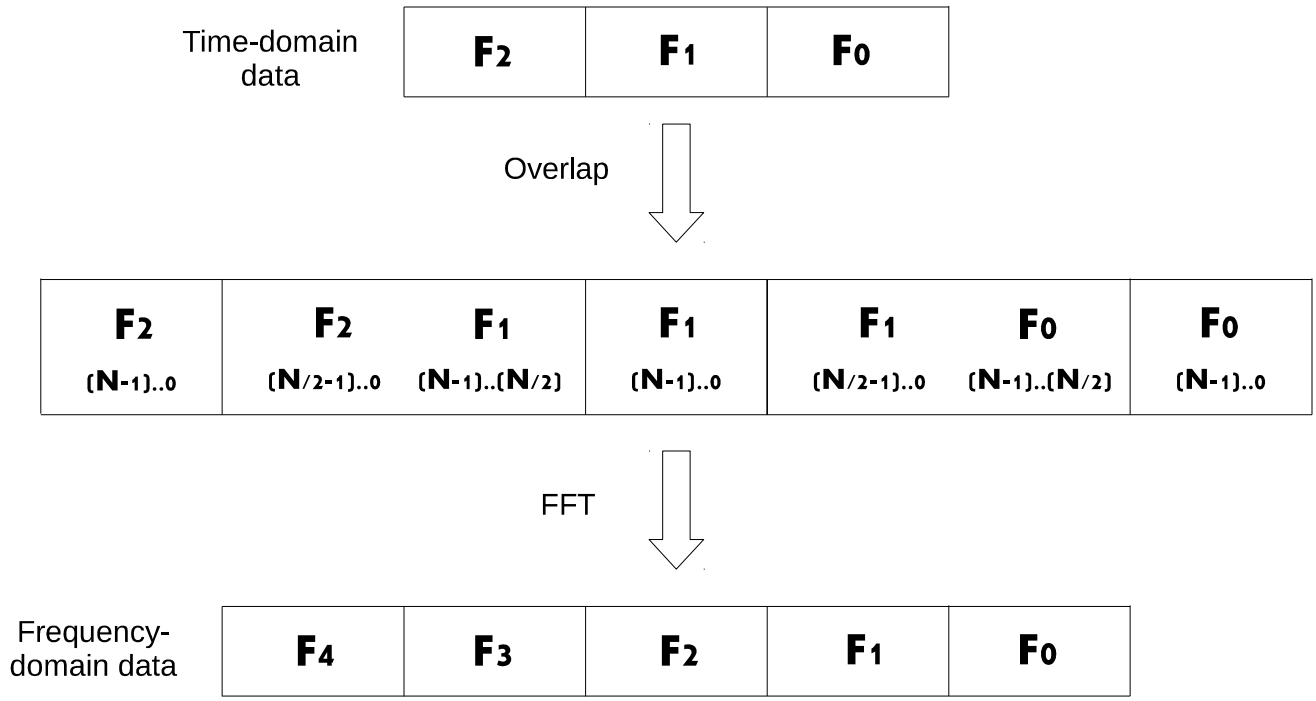
Disadvantages

- Done by CcmService. All unused data is sent over USB increasing the USB load
- The interval between frame buffers cannot be determined exactly

FFT Overlap

FFT results can sometimes be improved if a FFT block is generated using some of the raw time-domain data used to generate the previous block. The CCM has this capability built-in at the hardware level for use by the built-in FFT. This feature can be enabled or disabled using the API function `CcmSetFrequencyDomainOverlapEnabled`.

When enabled, a copy of the last 50% of the time-domain data used to generate the most recent FFT block becomes the first 50% of the time-domain block used to generate the next FFT block. This is shown in the following figure. N is the block size set using `CcmSetFrequencyDomainBlockSize`.



Note that enabling overlap effectively doubles the rate of the frequency-domain data that needs to be handled by the application, the service and the USB.

Camera control

The ACAM 100 has a built-in webcam in the center of the array. This allows an application to capture images of the sound sources that are being recorded by the microphones.

The images can be synchronized with the acoustic data, or the images can be captured asynchronously.

Synchronous capture

Synchronized capture requires a fixed temporal relation between the acoustic data and the optical images. This relation is specified using the function `CcmSetFramesPerImage(N)`. This initiates a protocol where exactly one image is sent from the camera to the API DLL in the middle of each set of N acoustic frames.

The application is required to read and process N acoustic frame and one image at each call to `CcmRead`.

Asynchronous capture

Asynchronous capture can be used where the environment does not change much. An image can be captured at any time using the function `CcmCaptureImage`. Any delay between the images and acoustic data depend on processing, buffering and OS delays.

Asynchronous capture is not available when synchronous capture is enabled using `CcmSetFramesPerImage(N)` with N > 0. If synchronous capture has been enabled, you must reset it by calling `CcmSetFramesPerImage(0)`.

Image control

The API has several functions for controlling the images:

`CcmSetCameraResolution`
: Change the size of the images returned to one of the options supported by the camera

`CcmSetImageFormat`
: Change the format of returned images to JPG, BMP, TIF, PNG or OpenCV Matrix

`CcmSetImageColorMode`
: Switch between color or black & white images

`CcmSetImageBrightness`
: Vary the brightness of the returned images

`CcmSetJpgCompression`
`CcmSetPngCompression`
: Change the level of image compression when possible

Service logging

The feature called Service Logging allows the CcmService to log incoming time-domain and/or frequency-domain data before sending it to the application. This can be used to record data for post-processing or to record data that would not otherwise be sent to the application when frame buffering is enabled.

The application can start and stop service logging data directly to disk by calling the API commands:

`CcmSetTimeDomainLoggingEnabled`
Start or stop time-domain logging

`CcmSetFrequencyDomainLoggingEnabled`
Start or stop frequency-domain logging

`CcmSetLoggingEnabled`
Simultaneously start or stop both time-domain and frequency-domain logging

Enabling logging before calling `CcmStartInput` causes logging to begin immediately when sampling starts. Since logging is done by the service, there is a small delay when starting or stopping when sampling is active.

Logging can be started, stopped and restarted at any time. For example, calling `CcmSetTimeDomainLoggingEnabled` to disable time-domain logging after starting with `CcmSetLoggingEnabled` will cause time-domain logging to stop but allow frequency-domain logging to continue. Calling `CcmStopInput` automatically stops all logging.

The data files are named to reflect the date and time of their creation.

A time-domain data file started on March 1, 2012; 3:32:47 PM would have the filename 120301_153247_TD.BIN

"_TD" is replaced by "_FD" for frequency-domain data files.

The files are placed in a directory named with the same date/time format. The previous file would be placed in the directory 120301_153247.

The parent of these data directories can be configured using the API function `CcmSetLogfileRoot`. If not specified, the default is C:\Data\Signal Interface Group. The fulll default path for the example above would be C:\Data\Signal Interface Group\120301_153247\120301_153247_TD.BIN.

When a new data folder is created, an XML configuration file describing the current state of the CCM is generated and placed in the new folder.

Existing binary files can be used by the application as a data source using the functions:

```
CcmSetTimeDomainSourceFile
CcmSetFrequencyDomainSourceFile
CcmGetTimeDomainSourceFile
CcmGetFrequencyDomainSourceFile
```

Configuration XML

After initializing the CCM system with the configuration settings used for a particular test, the state of the system can be saved to an XML file using the API function `CcmSaveConfiguration`. This file can be used to record the state of the system, or to reconfigure the system at a future time to the same state using the API function `CcmLoadConfiguration`.

The file contains all configurable and non-configurable settings including:

- Hardware and software version
- CCM serial number, description and model
- Camera and image settings
- Microphone calibration coefficients
- Sample Rate
- Block Size
- Skip and Take values
- Overlap on/off
- Frame Buffers on/off
- Frame Buffer size
- Microphone coordinates
- FFT Window type
- FFT Window coefficients

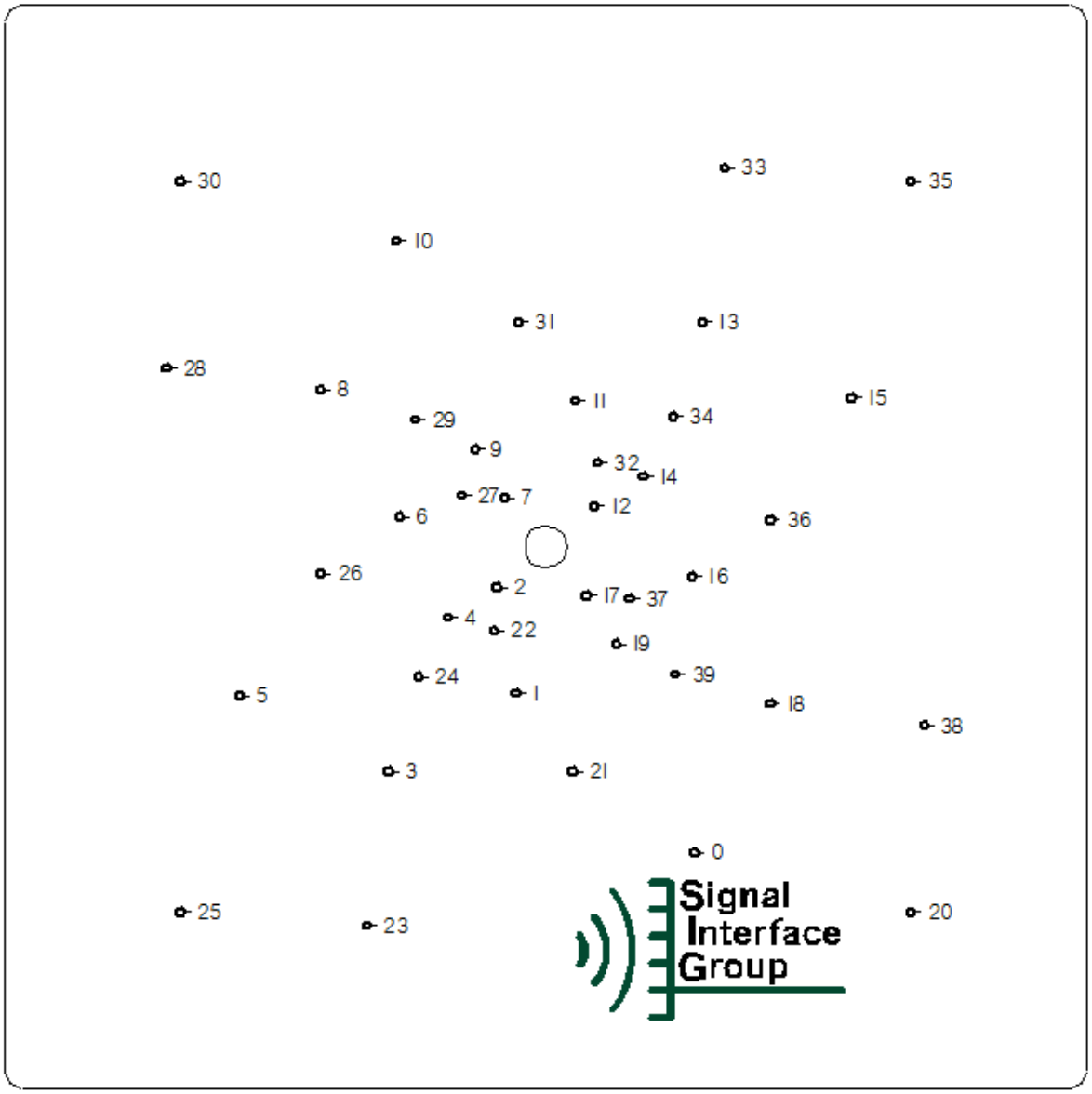
A full configuration example can be found in appendix C.

Microphone Locations

Knowing the locations of the microphones is critical for implementing a beamforming algorithm. The locations are stored in the array and can be obtained using the CcmApi function `CcmGetMicrophoneCoordinates`.

The coordinates are returned in millimeters, looking into the front of the array as shown in the picture below. The microphone numbering shown here matches the sample order for data returned from the array. The numbering on the internal PCB may be different from this.

Note that since the coordinates are referenced looking into the front of the array, the resulting beamform image will need to be mirrored. Alternatively, the application can change the sign of the X-coefficients obtained from `CcmGetMicrophoneCoordinates`.



Error Handling

Every API function returns an error code: CCM_ERR_NONE if successful or another error code if not successful. The function `CcmGetLastError` returns the last error code while `CcmGetLastErrorString` is used to obtain the corresponding text. An application must handle all errors appropriately.

Some of the following examples and code fragments omit error handling, but any realistic application must handle all possible errors, allowing for anything from invalid parameters to communication errors to a disconnected USB cable.

The error codes and error strings are listed in Appendix B.

Programming Examples

The examples in this chapter show how to use the API to read from a CCM system using C/C++ code. The same techniques apply to other languages, though the details depend on the language.

A realistic application has to do something interesting with the data: log the data to disk; graph the data; analyze the data and modify the outputs based on the values of the inputs. To keep the examples simple the data are just sent to the screen as text.

A typical application has these parts:

- gain access to I/O hardware
- acquire data
- clean up

In addiition to this generic example, simple working C++, Java and Visual Basic examples are provided in the Examples folder to get you started. Each one displays device information for the array and can be used to verify that the CcmAccess installation was successful.

Basic API Calls

`CcmOpen:`
Open the Command and Control Module for I/O.

`CcmStartInput:`
Starts input sampling.

`CcmRead:`
Read incoming time-domain and/or frequency-domain data and/or images

`CcmStopInput:`
Stops input sampling.

`CcmClose:`
Close all open connections to CCM hardware.

Example: Passing Data to a Spreadsheet or a Database

In many applications it is useful to log data to a text file in a format that is compatible with a spreadsheet or database. Often the preferred format is CSV: comma separated variables. The application writes the channel names in a row and then writes the data for each scan in a new row. Elements in each row are separated by commas.

This example writes a CSV file with a header row of channel names, then reads 10 frames from each channel and writes the data to the file.

In this example, the actual code which would be used to open, close and write to the file are replaced by the psdudo-code functions OpenFileX, CloseFileX and WriteFileX.

Note that since we want each row to be a scan (one sample from each microphone), the frame data must first be transposed. This is done by calling the function `CcmSetTimeDomainFrameOrder`.

Configuration and Cleanup

The following function shows a typical pattern for configuring a system and acquiring data.

```
void DataLogging(void)
{
   HANDLE file;
   int columns;

   /* for clarity, error handling is not shown */

   file = OpenFileX("example.csv");

   CcmOpen();

   CcmGetMicrophoneCount(&columns);
   GenerateColumnHeaders(columns);

   CcmSetSampleRate(CCM_SAMPLERATE_10000);
   CcmSetTimeDomainBlockSize(CCM_BLOCKSIZE_128);
   CcmSetTimeDomainEnabled(CCM_ENABLED);
   CcmSetTimeDomainFrameOrder(CCM_FRAMEORDER_SCAN);
   CcmSetFrequencyDomainEnabled(CCM_DISABLED);
   CcmStartInput();

   CollectData(columns, file);

   CcmStopInput();
   CcmClose();

   CloseFileX(file);
}
```

Description:

- initialize the logfile
- open USB communications (call `CcmOpen`)
- initialize device parameters
- set frame order to SCAN for data to arrive in column order
- start acquisition (call `CcmStartInput`)
- read, format and display data
- stop data acquisition (call `CcmStopInput`)
- close USB communications (call `CcmClose`)
- close the logfile

Reading Data

Read up to ten frames of time-domain data using `CcmRead`, and write to the file

At high sample rates, performance can be improved by reading multiple frames at each call to `CcmRead`, then separating them in the application code.

```
void CollectData(int columns, HANDLE file)
{
   int* buffer;
   char s[10];
   DWORD tdFrames, fdFrames;
   int framesize, blocksize;
   int frameCount = 0;
   int i;

   CcmGetTimeDomainBlockSize(&blocksize);

   // read one frame at a time
   framesize = blocksize * columns;
   buffer = new int[framesize];

   while (frameCount <= 10)
   {
      CcmRead(NULL,       // do NOT return freq-domain data
              0,
              NULL,
              buffer,     // DO return time-domain data
              framesize,
              &tdFrames,
              NULL,       // do NOT return images
              0,
              NULL,
              CCM_WAIT);

      // something is wrong if CCM_WAIT specified and no data returned
      if (tdFrames == 0)
         break;

      framecount++;

      for (i=0; i<columns; i++)
      {
            if (i < columns-1)
            sprintf(s, "%d,", buffer[i]);
         else
            sprintf(s, "%d\n", buffer[i]);

         WriteFileX(file, s);
      }
   }

   delete [] buffer;
}
```

Error Handling

All CcmApi functions return error values with the exception of `CcmGetLastErrorString` which returns nothing, and `CcmGetLastError` which returns the last error code.

Error values are type CCM_ERROR (DWORD), and can be found in the provided file CcmErrors.h. CCM_ERR_NONE (0) is returned if there is no error.

Example

```
void DataLogging(void)
{
   int channels;
   HANDLE file;
   char err[100];

   if (CcmOpen() != CCM_ERR_NONE)
   {
      CcmGetLastErrorString(err, 100);
      printf("Open fails with err %x (%s)\n", CcmGetLastError(), err);
   }

   /* ... the other functions are handled similarly  */

   if (CcmClose() != CCM_ERR_NONE)
   {
      CcmGetLastErrorString(err, 100);
      printf("Close fails with err %x (%s)\n", CcmLastError(), err);
   }
}
```

API Functions by Category

The following section provides a categorical listing of the CcmApi functions for quick reference. A separate alphabetical listing contains the complete details of each function can be found in appendix A.

Version Query

```
CcmGetDllVersion
CcmGetServiceVersion
CcmGetFirmwareVersion
```

Basic Status Query

```
CcmGetUsbConnectionStatus
CcmGetAvailabilityStatus
CcmGetDeviceHandleStatus
```

Device Information

```
CcmGetArrayDescription
CcmGetArrayModel
CcmGetArraySerialNumber
CcmGetMicrophoneCount
```

Device Access

```
CcmOpen
CcmClose
CcmSetPowerState
```

Basic I/O

```
CcmStartInput
CcmStopInput
CcmRead
CcmGetFrequencyDomainFramesAvailable
CcmGetTimeDomainFramesAvailable
```

Configuration

```
CcmGetSampleRate
CcmSetSampleRate

CcmGetTimeDomainBlockSize
CcmSetTimeDomainBlockSize
CcmGetTimeDomainEnabled
CcmSetTimeDomainEnabled
```

```
CcmGetFrequencyDomainBlockSize
CcmSetFrequencyDomainBlockSize
CcmGetFrequencyDomainEnabled
CcmSetFrequencyDomainEnabled
CcmGetFrequencyDomainWindowType
CcmSetFrequencyDomainWindowType
```

Camera Access

```
CcmCaptureImage

CcmGetCameraEnabled
CcmSetCameraEnabled
CcmGetCameraId
CcmSetCameraId
CcmGetFramesPerImage
CcmSetFramesPerImage
```

Image Settings

```
CcmGetCameraResolution
CcmSetCameraResolution
CcmSetImageBrightness
CcmGetImageBrightness
CcmSetImageColorMode
CcmGetImageColorMode
CcmGetImageFormat
CcmSetImageFormat
CcmGetJpgCompression
CcmSetJpgCompression
CcmGetPngCompression
CcmSetPngCompression

CcmGetImageExtents
CcmGetImageSize
```

Service Logging

```
CcmStartLogging
CcmStartTimeDomainLogging
CcmStartFrequencyDomainLogging

CcmStopLogging
CcmStopTimeDomainLogging
CcmStopFrequencyDomainLogging

CcmGetTimeDomainLoggingEnabled
CcmGetFrequencyDomainLoggingEnabled

CcmGetLogfileRoot
CcmSetLogfileRoot
```

Binary Source File

```
CcmSetTimeDomainSourceFile
CcmGetTimeDomainSourceFile

CcmSetFrequencyDomainSourceFile
CcmGetFrequencyDomainSourceFile
```

Status

```
CcmGetStatus
CcmGetMemoryPercentUsed
```

XML Configuration

```
CcmLoadConfiguration
CcmSaveConfiguration
```

Error Handling

```
CcmGetLastError
CcmGetLastErrorString
```

Frame Buffering

```
CcmGetTimeDomainFrameBufferSize
CcmSetTimeDomainFrameBufferSize
CcmGetTimeDomainFrameBufferAvailable

CcmSetFrequencyDomainFrameBufferSize
CcmGetFrequencyDomainFrameBufferSize
CcmGetFrequencyDomainFrameBufferAvailable
```

Beamforming Support

```
CcmGetMicrophoneCoordinates
```

Advanced Configuration

```
CcmSetFrequencyDomainCustomWindowCoefficients
CcmGetFrequencyDomainOverlapEnabled
CcmSetFrequencyDomainOverlapEnabled

CcmGetFrequencyDomainSkipFrames
CcmSetFrequencyDomainSkipFrames
CcmGetFrequencyDomainTakeFrames
CcmSetFrequencyDomainTakeFrames

CcmGetTimeDomainSkipFrames
CcmSetTimeDomainSkipFrames
CcmGetTimeDomainTakeFrames
CcmSetTimeDomainTakeFrames
```

Appendix A: API Function Calls

Appendix A is an alphabetical listing of the API function with examples.

Unless otherwise specified, all functions return an error code, or 0 (CCM_ERR_NONE) if no error occurred.

CcmCaptureImage

Description

Retrieve an image from the CCM camera

Syntax

```
DWORD CcmCaptureImage(BYTE* image, DWORD bufferBytes, DWORD* imageBytes);
```

Parameters

image
BYTE buffer which will receive the new image

bufferBytes
size of the image buffer in bytes. This declares the largest image that can be returned.

imageBytes
actual size of the returned image, in bytes

Notes

If synchronized images are configured using the function `CcmSetFramesPerImage`, the function will return the error `CCM_ERR_CAMERA_ASYNC_NOT_ALLOWED`.

If the buffer is too small to contain the image, the function will return the error `CCM_ERR_BUFFER_TOO_SMALL`. In this case, the image will remain in memory allowing the function to be called again with a larger buffer.

The function `CcmGetImageSize` returns the maximum image size in bytes. Using this value to allocate the image buffer will ensure a large enough buffer for all images.

Example

```
DWORD bufSize;
BYTE* image;
DWORD imageSize;
DWORD rc;

CcmGetImageSize(&bufSize);
image = new BYTE[bufSize];

rc = CcmCaptureImage(image, bufSize, &imageSize);
if (rc != CCM_ERR_NONE)
   ShowError(rc);
else
   ShowImage(image, imageSize);
```

CcmClose

Description

Close any I/O handles previously opened with CcmOpen

Syntax

```
DWORD CcmClose(void);
```

Parameters

none

Notes

Example

```
rc = CcmClose();
if (rc != CCM_ERR_NONE)
    // display error message
```

CcmGetArrayDescription

Description

Return the manufacturer's description of the array

Syntax

```
DWORD CcmGetArrayDescription(char* desc, DWORD length);
```

Parameters

desc
char buffer to receive the description text

length
size of the desc buffer

Notes

The array must be previously opened using CcmOpen before reading the description

Example

```
char desc[128];

if (CcmGetArrayDescription(desc, 128) == CCM_ERR_NONE)
    printf("description: %s\n", desc);
else
    printf("error: code=%d\n", CcmGetLastError());
```

CcmGetArrayModel

Description

Return the name of the array model

Syntax

```
DWORD CcmGetArrayModel(char* model, DWORD length);
```

Parameters

model
char buffer to receive the model text

length
size of the model buffer

Notes

The Array must be previously opened using CcmOpen before reading the model

Example

```
char model[128];

rc = CcmGetArrayModel(model, 128);

if (rc == CCM_ERR_NONE)
    printf("model: %s\n", model);
else
    printf("error: code=%d\n", CcmGetLastError());
```

CcmGetArraySerialNumber

Description

Return the serial number of the Acam 100 array

Syntax

```
DWORD CcmGetArraySerialNumber(char* sn, DWORD length);
```

Parameters

sn
char buffer to receive the serial number text

length
size of the sn buffer

Notes

The CCM must be previously opened using CcmOpen before reading the serial number

Example

```
char sn[128];

rc = CcmGetArraySerialNumber(sn, 128);

if (rc == CCM_ERR_NONE)
    printf("serial number: %s\n", sn);
else
    printf("error: %d\n", CcmGetLastError());
```

CcmGetAvailabilityStatus

Description

Returns whether or not the CCM device has been detected and properly initialized.

Syntax

```
DWORD CcmGetAvailabilityStatus(DWORD* isAvailable);
```

Parameters

isAvailable

upon successful return, value is set to `CCM_ENABLED` if the CCM device is available, or `CCM_DISABLED` if not.

Notes

Availability depends on a successful USB connection.

Example

```
DWORD isAvailable;
DWORD isConnected;
DWORD isOpened;

CcmGetUsbConnectionStatus(&isConnected);
if (isConnected == 0)
    printf("USB is not connected\n");
else
{
    CcmGetAvailabilityStatus(&isAvailable);
    if (isAvailable == 0)
        printf("Unable to detect CCM device\n");
    else
    {
        CcmGetDeviceHandleStatus(&isOpened);
        if (isOpened == 0)
           printf("Unable to open device for communication\n");
        else
           printf("CCM device is opened and ready for I/O!\n");
    }
}
```

CcmGetCameraEnabled

Description

Returns whether or not the CCM camera is available to capture images.

Syntax

```
DWORD CcmGetCameraEnabled(DWORD* isEnabled);
```

Parameters

isEnabled
upon successful return, value is set to `CCM_ENABLED` if the CCM camera is available, or `CCM_DISABLED` if not.

Notes

If enabling the camera without first setting the ID using `CcmSetCameraId`, the service will attempt to discover the ID. If unable to find a valid ID, or if the one provided by `CcmSetCameraId` is not a valid camera ID, the function will return with the error `CCM_ERR_CAMERA_NOT_OPENED`.

Disabling the camera is equivalent to calling the function `CcmSetCameraId(CCM_INVALID_CAMERA_ID);`

Example

```
DWORD isEnabled;

CcmGetCameraEnabled(&isEnabled);
if (isEnabled == CCM_ENABLED)
    printf("The camera is enabled\n");
else
{
    DWORD rc = CcmSetCameraEnabled(CCM_ENABLED);
    if (rc == CCM_ERR_CAMERA_NOT_OPENED))
       printf("Camera not available\n");
}
```

CcmGetCameraId

Description

Returns the current camera ID

Syntax

```
DWORD CcmGetCameraId(DWORD* ID);
```

Parameters

ID

upon successful return, value is the current webcam ID, or CCM_INVALID_CAMERA_ID if the camera is unavailable or not initialized

Notes

The CcmService automatically detects the ID of the CCM's onboard webcam when the CCM is plugged into the USB port. A different ID may be selected for an external camera.

Example

```
DWORD id;

CcmGetCameraId(&id);
if (id == CCM_INVALID_CAMERA_ID)
    printf("The camera is not available or has not been selected.\n");
else
    printf("Camera ready!\n");
```

CcmGetCameraResolution

Description

Returns the current webcam resolution

Syntax

```
DWORD CcmGetCameraId(DWORD* resolutionCode);
```

Parameters

resolutionCode

upon successful return, value is a code that represents the resolution. The available codes are found in the file CcmApi.h. They are:

```
CCM_IMAGESIZE_1024x768
CCM_IMAGESIZE_800x600
CCM_IMAGESIZE_640x480
CCM_IMAGESIZE_320x240
```

Notes

The function `CcmGetImageExtents` can be used to return the horizontal and vertical resolutions as separate values.

Example

```
DWORD res;
DWORD x, y;

CcmGetCameraResolution(&res);

switch(res)
{
   case CCM_IMAGESIZE_1024x768:
      x=1024; y=768; break;

   case CCM_IMAGESIZE_800x600:
      x=800; y=600; break;

   case CCM_IMAGESIZE_640x480:
      x=640; y=480; break;

   case CCM_IMAGESIZE_320x240:
      x=320; y=240; break;

   default:
      printf("Invalid res code returned: %d\n", res);
      return;
}

printf("Resolution: horiz=%d, vert=%d\n", x, y);
```

CcmGetCurrentLogfileFolder

Description

Returns the folder currently being logged to.

Syntax

```
DWORD CcmGetCurrentLogfileFolder(char* path, DWORD maxLength);
```

Parameters

path

upon successful return, *path* contains the name of the folder in which logfiles are currently being written to. If no logfiles are currently being written to, *path* contains the most recent folder used for logging. If no logfiles have been written, *path* is empty.

maxLength

the size of the path buffer which determines the maximum size of the returned string

Notes

none

Example

```
char path[256];

CcmStartLogging();
CcmGetCurrentLogfileFolder(path, 256);

printf("Data currently being logged to folder: %s\n", path);
```

CcmGetDeviceHandleStatus

Description

Returns whether or not the Array's device handle has been previously opened using CcmOpen.

Syntax

```
DWORD CcmGetDeviceHandleStatus(DWORD* isOpened);
```

Parameters

isOpened

on successful return, value is set to 1 if the device handles are opened, or 0 if not.

Notes

Device cannot be opened if USB is disconnected

Example

```
DWORD isAvailable;
DWORD isConnected;
DWORD isOpened;

CcmGetUsbConnectionStatus(&isConnected);
if (isConnected == 0)
    printf("USB is not connected\n");
else
{
    CcmGetAvailabilityStatus(&isAvailable);
    if (isAvailable == 0)
        printf("Unable to detect the array\n");
    else
    {
        CcmGetDeviceHandleStatus(&isOpened);
        if (isOpened == 0)
           printf("Unable to open device for communication\n");
        else
           printf("CCM device is opened and ready for I/O!\n");
    }
}
```

CcmGetDllVersion

Description

Return the version of the DLL being used

Syntax

```
DWORD CcmGetDllVersion(DWORD* major, DWORD* minor, DWORD* build);
```

Parameters

major, minor, build
pointers to 32-bit unsigned integers that receives the version information

Example

```
DWORD minor, major, build;

if (CcmGetDllVersion(&major, &minor, &build) > CCM_ERR_NONE)
    printf("Error reading DLL version: %d\n", CcmGetLastError());
else
    printf("major=%d, minor=%d, build=%d\n", major, minor, build);
```

CcmGetFirmwareVersion

Description

Return the firmware version of the current CCM

Syntax

```
DWORD CcmGetFirmwareVersion(DWORD* major, DWORD* minor, DWORD* build);
```

Parameters

major, minor, build
pointers to 32-bit unsigned integers that receives the version information

Example

```
DWORD minor, major, build;

if (CcmGetFirmwareVersion(&major, &minor, &build) > CCM_ERR_NONE)
    printf("Error reading firmware version: %d\n", CcmGetLastError());
else
    printf("major=%d, minor=%d, build=%d\n", major, minor, build);
```

CcmGetFramesPerImage

Description

Return the current "frames per image" setting used for synchronized camera images.

Syntax

```
DWORD CcmGetFramesPerImage(DWORD* fpi);
```

Parameters

fpi
pointers to 32-bit unsigned integer that receives the current Frames Per Image setting

Notes

If this value is non-zero, one image is automatically captured for every *fpi* acoustic frames. The application is required to read *fpi* frames and one image at each call to `CcmRead`.

If the value is zero, images are not automatically returned but can be requested using the function `CcmCaptureImage`.

Example

```
DWORD fpi;
DWORD rc;

rc = CcmGetFramesPerImage(&fpi);

if (rc == CCM_ERR_NONE)
   printf("Set read buffer for %d frames\n", fpi);
else
   printf("Error reading frames-per-image: %x\n", rc);
```

CcmGetFrequencyDomainBlockSize

Description

Get the value of the block size currently used with frequency-domain data.

Syntax

```
DWORD CcmGetFrequencyDomainBlockSize(DWORD* blockSize);
```

Parameters

blockSize
pointer to 32-bit unsigned integer that receives the block size value.

Example

```
DWORD blocksize;

CcmGetFrequencyDomainBlockSize(&blocksize);
printf("Frequency domain block size is %d\n", blocksize);
```

CcmGetFrequencyDomainEnabled

Description

Find out if the service is configured to return frequency domain data.

Syntax

```
DWORD CcmGetFrequencyDomainEnabled(DWORD* enabled);
```

Parameters

enabled

on successful return, value is set to CCM_ENABLED (1) if enabled, or CCM_DISABLED (0) if disabled.

Example

```
DWORD enabled;
DWORD size, framesize;
DWORD *tdBuffer=NULL, *fdBuffer=NULL;

framesize = GetFramesize();
size = 10 * framesize;

CcmGetFrequencyDomainEnabled(&enabled);
if (enabled)
    fdBuffer = new DWORD(size);

CcmGetTimeDomainEnabled(&enabled);
if (enabled)
    tdBuffer = new DWORD(size);

ReadData(tdBuffer, fdBuffer, size);
```

CcmGetFrequencyDomainFrameBufferAvailable

Description

Find out if the full frequency domain frame buffer is available for reading

Syntax

```
DWORD CcmGetFrequencyDomainFrameBufferAvailable(DWORD* avail);
```

Parameters

avail

on successful return, value is set to 1 if the full frame buffer is available, or 0 if not.

Example

```
DWORD enabled;
DWORD size;
DWORD avail = 0;

CcmGetFrequencyDomainEnabled(&enabled);

if (enabled == CCM_ENABLED)
{
    CcmGetFrequencyDomainFrameBufferSize(&size);

    if (size > 0)
        CcmGetFrequencyDomainFrameBufferAvailable(avail);
    else
        CcmGetFrequencyDomainFrameAvailable(avail);
}

if (avail > 0)
    ReadFrequencyDomainData();
```

CcmGetFrequencyDomainFrameBufferSize

Description

Get the number of frames maintained in the frequency domain frame buffer.

Syntax

```
DWORD CcmGetFrequencyDomainFrameBufferSize(DWORD* frames);
```

Parameters

frames

pointer to 32-bit unsigned integer that receives the number of frames

Notes

The frame buffer stores only the most recent frames and discards the older ones.

Frame buffering is enabled when the Frame Buffer Size > 0.

Example

```
DWORD frameCount;
DWORD micCount;
DWORD blocksize;
DWORD framesize;

CcmGetMicrophoneCount(&micCount);
CcmGetFrequencyDomainBlockSize(&blocksize);
CcmGetFrequencyDomainFrameBufferSize(&frameCount);

framesize = micCount * blocksize;
printf("Frame buffer holds %d samples\n", framesize*frameCount);
```

CcmGetFrequencyDomainFramesAvailable

Description

Return the number of full frequency domain frames available for reading

Syntax

```
DWORD CcmGetFrequencyDomainFramesAvailable(DWORD* frames);
```

Parameters

frames

on successful return, value is set to the number of full frequency domain frames available.

Example

```
DWORD fdFrames, tdFrames;
DWORD fdRead, tdRead;

CcmGetFrequencyDomainFramesAvailable(&fdFrames);
CcmGetTimeDomainFramesAvailable(&tdFrames);

if ((fdFrames > 0) && (tdFrames > 0))
{
    CcmRead(fdBuffer, size, &fdRead, tdBuffer, size, &tdRead,
            NULL, 0, NULL, CCM_NO_WAIT);

    ProcessData(tdBuffer, fdBuffer, tdRead, fdRead);
}
```

CcmGetFrequencyDomainLoggingEnabled

Description

Return the current frequency-domain service logging state.

Syntax

```
DWORD CcmGetFrequencyDomainLoggingEnabled(DWORD* enabled);
```

Parameters

enabled
pointer to the integer that enabled status (CCM_ENABLED or CCM_DISABLED)

Example

```
DWORD enabledFd, enabledTd;

CcmGetFrequencyDomainLoggingEnabled(&enabledFd);
CcmGetTimeDomainLoggineEnabled(&enabledTd);

if (enabledFd == CCM_ENABLED)
    printf("Frequency domain logging is enabled.\n");

if (enabledTd == CCM_ENABLED)
    printf("Time domain logging is enabled.\n");

if (enabledTd == CCM_DISABLED && enabledFd == CCM_DISABLED)
    printf("Logging disabled\n");
```

CcmGetFrequencyDomainOverlapEnabled

Description

Find out if time-domain overlap for FFT operation is enabled

Syntax

```
DWORD CcmGetFrequencyDomainOverlapEnabled(DWORD* enabled);
```

Parameters

enabled

on successful return, value is set to 1 if overlap is enabled, or 0 if not.

Notes

Overlap means reusing the last half of the previous time-domain data block in generating a block of frequency domain data.

Example

```
DWORD enabled;

// force overlap to be enabled if not already enabled

CcmGetFrequencyDomainOverlapEnabled(&enabled);
if (enabled == CCM_DISABLED)
    CcmSetFrequencyDomainOverlapEnabled(CCM_ENABLED);
```

CcmGetFrequencyDomainSkipFrames

Description

Get the number of frames that are skipped at the hardware level after taking a specified number of frames.

Syntax

```
DWORD CcmGetFrequencyDomainSkipFrames(DWORD* frames);
```

Parameters

frames
number of frames that are skipped

Notes

Skip must be used in conjunction with Take.

Example

```
DWORD skip, take, samplerate;
float datarate;

CcmGetFrequencyDomainSkipFrames(&skip);
CcmGetFrequencyDomainTakeFrames(&take);
CcmGetSampleRate(&samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("Effective data rate with skip and take = %f Hz\n", datarate);
```

CcmGetFrequencyDomainSourceFile

Description

Get the name of the current frequency-domain binary source file, if selected.

Syntax

```
DWORD CcmGetFrequencyDomainSourceFile(char* filename,
                                      DWORD size,
                                      DWORD* enabled);
```

Parameters

filename
char buffer that receives the currently configured binary source file

size
size of the filename buffer

enabled
set to `CCM_ENABLED` or `CCM_DISABLED` indicating whether or not the source file is open

Notes

On return, an empty filename shows that no file has been selected. Enabled can mean either that the filename has not been selected, or was not opened properly.

The function `CcmStopInput` closes the file and resets the filename to an empty string.

Example

```
DWORD enabled;
char filename[100];

CcmGetFrequencyDomainSourceFile(filename, 100, &enabled);

if (strlen(filename) == 0)
   printf("No file has been selected\n");
else if (enabled == CCM_DISABLED)
   printf("Error opening file: %s\n", filename);
else
   printf("File %s is ready\n", filename);
```

CcmGetFrequencyDomainTakeFrames

Description

Get the number of frames that are returned (taken) at the hardware level after skipping a specified number of frames.

Syntax

```
DWORD CcmGetFrequencyDomainTakeFrames(DWORD* frames);
```

Parameters

frames
number of frames that are taken

Notes

Take must be used in conjunction with Skip.

Example

```
DWORD skip, take, samplerate;
float datarate;

CcmGetFrequencyDomainSkipFrames(&skip);
CcmGetFrequencyDomainTakeFrames(&take);
CcmGetSampleRate(&samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("effective data rate with skip and take = %f\n", datarate);
```

CcmGetFrequencyDomainWindowType

Description

Get the value of the current FFT window type.

Syntax

```
DWORD CcmGetFrequencyDomainWindowType(DWORD* window);
```

Parameters

window

Pointer to variable that receives current FFT window type. The possibilities are:

CCM_FFT_WINDOW_RECTANGULAR
CCM_FFT_WINDOW_HAMMING
CCM_FFT_WINDOW_HANNING
CCM_FFT_WINDOW_CUSTOM

Example

```
/* generate an FFT window and save with the configuration */

CcmGetFrequencyDomainWindowType(&lastWindow);
CcmSetFrequencyDomainCustomWindowCoefficients(buffer, blockSize/2, &written);
CcmSetFrequencyDomainWindowType(CCM_WINDOW_CUSTOM);
CcmSaveConfiguration("customWindow.cfg");
```

CcmGetImageBrightness

Description

Get the current brightness setting of the camera.

Syntax

```
DWORD CcmGetImageBrightness(DWORD* brightness);
```

Parameters

brightness
pointer to integer that receives the brightness value

Note

The range of the brightness level is determined by the camera. The current array has a range of 0 to 15.

Example

```
DWORD brightness;

CcmGetImageBrightness(&brightness);
printf("The current brightness level is %d\n", brightness);
```

CcmGetImageColorMode

Description

Get the current color mode of the returned images

Syntax

```
DWORD CcmGetImageColorMode(DWORD* code);
```

Parameters

code

one of the possible color mode codes found in CcmApi.h:

CCM_IMAGECOLOR_RGB24:	24-bit color (default)
CCM_IMAGECOLOR_BW8:	8-bit black and white

Note

Reducing the color depth to 8-bit black and white decreases the file size of uncompressed images by a factor of three, but may not significantly decrease the size of compressed images.

Example

```
DWORD code;

CcmGetImageColorMode(&code);

switch(code)
{
   case CCM_IMAGECOLOR_RGB24:
      printf("Images are 24-bit color\n");
      break;

   case CCM_IMAGECOLOR_BW8:
      printf("Images are 8-bit black and white\n");
      break;

   default
      printf("Error: unknown color code: %d\n", code);
}
```

CcmGetImageFormat

Description

Get the format of the images returned by the API

Syntax

```
DWORD CcmGetImageFormat(DWORD* fmt);
```

Parameters

fmt

one of the possible format codes found in CcmApi.h:

`CCM_IMAGEFORMAT_JPG` – default
`CCM_IMAGEFORMAT_BMP`
`CCM_IMAGEFORMAT_PNG`
`CCM_IMAGEFORMAT_TIF`
`CCM_IMAGEFORMAT_MAT` – returns data part of OpenCv matrix

Note

With the exception of `CCM_IMAGEFORMAT_MAT`, each returned image is complete - that is, the buffer can be stored to disk and opened with an appropriate viewer. No extra processing of the image data is required.

Example

```
DWORD code=0;
DWORD size=0;
BYTE* image = new BYTE[100000];

if (CcmCaptureImage(image, 100000, &size) == CCM_ERR_BUFFER_TOO_SMALL)
{
   printf("Image is too large for buffer\n");
   delete [] image;
   return;
}

CcmGetImageFormat(code);

switch(code)
{
   case CCM_IMAGEFORMAT_JPG: ShowJpgImage(image, size); break;
   case CCM_IMAGEFORMAT_BMP: ShowBmpImage(image, size); break;
   case CCM_IMAGEFORMAT_PNG: ShowPngImage(image, size); break;
   case CCM_IMAGEFORMAT_TIF: ShowTifImage(image, size); break;
   case CCM_IMAGEFORMAT_MAT: ProcessMat(image, size); break;
   default: printf("Invalid image type: %d\n", code);
}

delete [] image;
```

CcmGetImageSize

Description

Get the maximum size of the images to be returned

Syntax

```
DWORD CcmGetImageSize(DWORD* maxBytes);
```

Parameters

maxBytes

the maximum size of the images to be returned give the current set of image parameters: resolution, color mode, image type and compression level.

Note

This function is used to determine the size of the of the image buffer to be allocated. It must be called after the resolution, color mode and image type, and compression level are set.

For compressed images, the size is based on a random-generated image with low compressability. The actual size of the images returned are typically much smaller.

Example

```
DWORD code=0;
DWORD size=0;
DWORD bufsize=0;
BYTE* image;

CcmGetImageSize(&bufsize);
if (bufsize == 0)
{
   printf("Unable to determine maximum image size\n");
   return;
}

image = new BYTE[bufsize];

CcmCaptureImage(image, bufsize, &size);
ShowImage();

delete [] image;
```

CcmGetJpgCompression

Description

Get the current level of JPG image compression

Syntax

```
DWORD CcmGetJpgCompression(DWORD* compression);
```

Parameters

compression

the current compression level:

0 = maximum compression (smallest file size, poor quality)
100 = minimum compression (largest file size, higher quality)

The default JPG compression level is 95

Example

```
DWORD comp;

CcmGetJpgCompression(&comp);
SetJpgCompressionSlider(comp);
```

CcmGetLastError

Description

Get the last error code that was generated by the CcmApi DLL.

Syntax

```
DWORD CcmGetLastError(void);
```

Parameters

none

Return Value

Funciton returns the last error code.

Note

Most of the CcmApi functions return the last error code so this function is rarely needed.

Example

```
if (CcmOpen() > 0)
   printf("CcmOpen fails with error: 0x%x", CcmGetLastError());
```

CcmGetLastErrorString

Description

Get the text of the last error that was generated by the CcmApi DLL.

Syntax

```
DWORD CcmGetLastErrorString(char* errorMsg, DWORD length);
```

Parameters

errorMsg
char buffer used to return the null-terminated error string

length
the size of *errorMsg,* the maximum length of the error string, including null-terminator

Note

The maximum length of the error text is 256 characters, including the null-terminator

Example

```
char err[100];

CcmOpen();
if (CcmGetLastError() != CCM_ERR_NONE)
{
    CcmGetLastErrorString(err, 100);
    printf("Error: %s\n", err);
}
```

CcmGetLogfileRoot

Description

Gets the current root folder used for server logging

Syntax

```
DWORD CcmGetLogfileRoot(char* path);
```

Parameters

path
null-teriminated string of the new base folder

Notes

Logfiles are written to subfolders within the logfile root folder. These subfolders have names based on the date and time that loggine was started.

If path returns with an empty string, the current path is the default path "C:\Data\Signal Interface Group".

Example

```
CcmGetLogfileRoot(path);
printf("logging to %s\n", path);
```

CcmGetMemoryPercentUsed

Description

Determine how much of the internal buffer memory is used, indicating how long before an overflow occurs. An overflow means that there is no room to put new data, and input sampling is stopped.

Syntax

```
DWORD CcmGetMemoryPercentUsed(DWORD* percentUsed);
```

Parameters

percentUsed
pointer to the integer receiving the amount of buffer memory that is used, in percentage of total buffer memory

Notes

After an overflow occurs and input sampling is stopped, the application can continue reading buffered data until the buffer is empty.

Example

```
DWORD pctUsed;

CcmGetPercentMemoryUsed(&pctUsed);

if (pctUsed < 10)
    printf("buffer almost empty\n");
else if (pctUsed > 45 && pctUsed < 55)
    printf("buffer about half full\n");
else if (pctUsed > 90)
    printf("buffer almost full, overflow imminent");
```

CcmGetMicrophoneCoordinates

Description

Retrieve the coordinates, in millimeters, of the microphones on the microphone PCB.

Syntax

```
DWORD CcmGetMicrophoneCoordinates(float* coordsX, float* coordsY,
                                  DWORD  sizeX,   DWORD  sizeY,
                                  DWORD* countX,  DWORD* countY)
```

Parameters

coordsX, coordsY
32-bit floating-point arrays that will receive the X and Y coordinates

sizeX, sizeY
maximum number of coordinates to return to each array

countX, countY
pointers to the integers receiving the number of X and Y coordinates returned

Notes

Coordinates are referenced to the center of the array.

Coordinates are stored so that, when viewing the array from the front with the Signal Interface Group logo properly oriented, the location of the first microphone sampled is the first coordinate pair.

sizeX and sizeY parameters must be the same.

Example

```
float* coordsX;
float* coordsY;
DWORD micCount;
DWORD coordCountX, coordCountY;

CcmGetMicrophoneCount(&micCount);
coordsX = new DWORD(micCount);
coordsY = new DWORD(micCount);

CcmGetMicrophoneCoordinates(coordsX, coordsY, micCount, micCount,
&coordCountX, &coordCountY);

for (int i=0; i<coordCountX; i++)
    printf("Mic[%d]: \tx=%f, \ty=%f\n", i, coordsX[i], coordsY[i];
```

CcmGetMicrophoneCount

Description

Get the number of microphones supported by the current CCM device.

Syntax

```
DWORD CcmGetMicrophoneCount(DWORD* micCount);
```

Parameters

micCount
pointer to the integer that receives the number of microphones

Example

```
DWORD* coords;
DWORD micCount;
DWORD coordCount;
float x, y;

CcmGetMicrophoneCount(&micCount);
coords = new DWORD(micCount);

CcmGetMicrophoneCoordinates(coords, micCount, &coordCount);

for (int i=0; i<coordCount; i++)
{
    x = (float)(coords[i] >> 16) / 10.0;
    y = (float)(coords[i] & 0xFFFF0000) / 10.0;
    printf("Mic[%d]: x=%f, y=%f\n", i, x, y);
}
```

CcmGetPngCompression

Description

Get the current level of PNG image compression

Syntax

```
DWORD CcmGetPngCompression(DWORD* compression);
```

Parameters

compression

the current compression level:

0 = minimum compression effort (larger file size, shorter compression time)
9 = maximum compression effort (smaller file size, longer compression time)

The default PNG compression level is 3

Notes

PNG is a lossless compression, so images have the same quality regardless of the compression level selected. The only effect is on the file size. Typically, the difference in file size between moderate compression effort and and high compression effort is minimal.

Example

```
DWORD comp;

CcmGetPngCompression(&comp);
SetPngCompressionSlider(comp);
```

CcmGetSampleRate

Description

Return the current sample rate in Hz

Syntax

```
DWORD CcmGetSampleRate(DWORD* rate);
```

Parameters

rate
pointer to integer that receives the sample rate

Example

```
DWORD rate;

if (CcmGetSampleRate(&rate) != CCM_ERR_NONE)
    printf("Unable to read sample rate: err = %d\n", CcmGetLastError());
else
    printf("Sample rate = %d Hz\n", rate);
```

CcmGetSampleRateList

Description

Get the list of sample rates supported by the Array

Syntax

```
DWORD CcmGetSampleRateList(DWORD* list, DWORD maxsize, DWORD* count);
```

Parameters

list
: array that receives the list of sample rates

maxsize
: size of the array

count
: upon return, the number of items in the array

Notes

This function is by an application to populate a selection menu of available sample rates.

The set of sample rates is dependent on the Array's communication and controller module (CCM) hardware.

Example

```
DWORD rates[10];
DWORD count;

CcmGetSampleRateList(rates, 10, &count);

for (int i=0; i<count; i++)
   printf("RATE %d = %d Hz\n", i, rates[i]);
```

CcmGetServiceVersion

Description

Return the version of the service being used

Syntax

```
DWORD CcmGetServiceVersion(DWORD* major, DWORD* minor, DWORD* build);
```

Parameters

major, minor, build
pointer to 32-bit unsigned integers that receives the version information

Example

```
DWORD minor, major, build;

if (CcmGetServiceVersion(&major, &minor, &build) > CCM_ERR_NONE)
    printf("Error reading service version: %d\n", CcmGetLastError());
else
    printf("major=%d, minor=%d, build=%d\n", major, minor, build);
```

CcmGetStatus

Description

Return the current status information

Syntax

```
DWORD CcmGetStatus(DWORD* status, DWORD* valid);
```

Parameters

status
pointer to 32-bit unsigned integer that receives the current status information

valid
indicates the validity of the status. If zero, status has not been updated since the last call to CcmGetStatus. Non-zero means the status has been updated. The status might not be updated if the function is called too often, or if the internal data buffers have overflowed.

Notes

The status is a 32-bit field with the bit values defined in CcmApi.h as follows:

```
CCM_STATUS_SAMPLING    0x00000001
CCM_STATUS_OVERFLOW    0x00000002
CCM_STATUS_VALID_MASK  0x00001000
CCM_STATUS_LOGGING_ENA 0x00002000
CCM_STATUS_MEMORY_FULL 0x00008000
```

Other bits may be active, but are not generally useful.

Example

```
DWORD status, valid;

CcmGetStatus(&status, &valid);

if (!valid)
    printf("No new status\n");
else
    printf("Input sampling %s\n",
       (status & CCM_STATUS_SAMPLING) ? "active" : "not active");
```

CcmGetTimeDomainBlockSize

Description

Get the value of the block size currently used with time-domain data.

Syntax

```
DWORD CcmGetTimeDomainBlockSize(DWORD* blockSize);
```

Parameters

blockSize
pointer to 32-bit unsigned integer that receives the block size value.

Example

```
DWORD blocksize;

CcmGetTimeDomainBlockSize(&blocksize);
printf("Time domain block size is %d\n", blocksize);
```

CcmGetTimeDomainEnabled

Description

Find out if the service is configured to return time domain data.

Syntax

```
DWORD CcmGetTimeDomainEnabled(DWORD* enabled);
```

Parameters

enabled

on successful return, value is set to CCM_ENABLED or CCM_DISABLED.

Example

```
DWORD enabled;
DWORD size, framesize;
DWORD *tdBuffer=NULL, *fdBuffer=NULL;

framesize = GetFramesize();
size = 10 * framesize;

CcmGetFrequencyDomainEnabled(&enabled);
if (enabled == CCM_ENABLED)
    fdBuffer = new DWORD(size);

CcmGetTimeDomainEnabled(&enabled);
if (enabled == CCM_ENABLED)
    tdBuffer = new DWORD(size);

ReadData(tdBuffer, fdBuffer, size);
```

CcmGetTimeDomainFrameBufferAvailable

Description

Find out if the full time domain frame buffer is available for reading

Syntax

```
DWORD CcmGetTimeDomainFrameBufferAvailable(DWORD* avail);
```

Parameters

avail

on successful return, value is set to 1 if a full frame buffer is available, or 0 if not.

Example

```
DWORD enabled;
DWORD size;
DWORD avail = 0;

CcmGetTimeDomainEnabled(&enabled);

if (enabled)
{
    CcmGetTimeDomainFrameBufferSize(&size);

    if (size > 0)
        CcmGetTimeDomainFrameBufferAvailable(avail);
    else
        CcmGetTimeDomainFramesAvailable(avail);
}

if (avail > 0)
    ReadTimeDomainData();
```

CcmGetTimeDomainFrameBufferSize

Description

Get the number of frames maintained in the time domain frame buffer.

Syntax

```
DWORD CcmGetTimeDomainFrameBufferSize(DWORD* frames);
```

Parameters

frames
pointer to 32-bit unsigned integer that receives the number of frames

Notes

The frame buffer stores only the most recent frames and discards the older ones.

Example

```
DWORD frameCount;
DWORD micCount;
DWORD blocksize;
DWORD framesize;

CcmGetMicrophoneCount(&micCount);
CcmGetTimeDomainBlockSize(&blocksize);
CcmGetTimeDomainFrameBufferSize(&frameCount);

framesize = micCount * blocksize;
printf("Frame buffer holds %d samples\n", framesize*frameCount);
```

CcmGetTimeDomainFrameOrder

Description

Determine if the time domain data is returned in block order (default) or scan order (transpose).

Syntax

```
DWORD CcmGetTimeDomainFrameOrder(DWORD* order);
```

Parameters

order

pointer to 32-bit unsigned integer that receives the frame order:

`CCM_FRAMEORDER_BLOCK` (default)
`CCM_FRAMEORDER_SCAN` (transpose)

Notes

`CCM_FRAMEORDER_SCAN` combined with CcmRead has the same functionality as `CCM_TRANSPOSE` did with the deprecated CcmReadFrames function.

Example

```
DWORD order;

CcmGetTimeDomainFrameOrder(&order);

if (order == CCM_FRAMEORDER_BLOCK)
   printf("Change frame order to SCAN before creating data table\n");
else
   CreateTable();
```

CcmGetTimeDomainFramesAvailable

Description

Return the number of full time domain frames available for reading

Syntax

```
DWORD CcmGetTimeDomainFramesAvailable(DWORD* frames);
```

Parameters

frames

on successful return, value is set to the number of full time domain frames available for reading

Example

```
DWORD fdFrames, tdFrames;
DWORD fdRead, tdRead;

CcmGetFrequencyDomainFramesAvailable(&fdFrames);
CcmGetTimeDomainFramesAvailable(&tdFrames);

if ((fdFrames > 0) && (tdFrames > 0))
{
    CcmRead(fdBuffer, size, &fdRead, tdBuffer, size, &tdRead,
            NULL, 0, NULL, CCM_NO_WAIT);

    ProcessData(tdBuffer, fdBuffer, tdRead, fdRead);
}
```

CcmGetTimeDomainLoggingEnabled

Description

Detemine whether service-level time domain logging is enabled.

Syntax

```
DWORD CcmGetTimeDomainLoggingEnabled(DWORD* enabled);
```

Parameters

enabled
pointer to the integer that enabled status (CCM_ENABLED or CCM_DISABLED)

Example

```
DWORD enabledFd, enabledTd;

CcmGetFrequencyDomainLoggingEnabled(&enabledFd);
CcmGetTimeDomainLoggineEnabled(&enabledTd);

if (enabledFd == CCM_ENABLED)
    printf("Frequency domain logging is enabled.\n");

if (enabledTd == CCM_ENABLED)
    printf("Time domain logging is enabled.\n");

if (enabledTd == CCM_DISABLED && enabledFd == CCM_DISABLED)
    printf("Logging disabled\n");
```

CcmGetTimeDomainSkipFrames

Description

Get the number of frames that are skipped at the hardware level after taking a specified number of frames.

Syntax

```
DWORD CcmGetTimeDomainSkipFrames(DWORD* frames);
```

Parameters

frames
number of frames that are skipped

Notes

Skip must be used in conjunction with Take.

Example

```
DWORD skip, take, samplerate;
float datarate;

CcmGetTimeDomainSkipFrames(&skip);
CcmGetTimeDomainTakeFrames(&take);
CcmGetSampleRate(samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("Effective data rate with skip and take = %f Hz\n", datarate);
```

CcmGetTimeDomainSourceFile

Description

Get the name of the current time-domain binary source file, if selected.

Syntax

```
DWORD CcmGetTimeDomainSourceFile(char* filename, DWORD size, DWORD* enabled);
```

Parameters

filename
char buffer that receives the currently configured binary source file

size
size of the filename buffer

enabled
set to `CCM_ENABLED` or `CCM_DISABLED` indicating whether or not the source file is open

Notes

On return, an empty filename shows that no file has been selected. Enabled can mean either that the filename has not been selected, or was not opened properly.

The function `CcmStopInput` closes the file and resets the filename to an empty string.

Example

```
DWORD enabled;
char filename[100];

CcmGetTimeDomainSourceFile(filename, 100, &enabled);

if (strlen(filename) == 0)
   printf("No file has been selected\n");
else if (enabled == CCM_DISABLED)
   printf("Error opening file: %s\n", filename);
else
   printf("File %s is ready\n", filename);
```

CcmGetTimeDomainTakeFrames

Description

Get the number of frames that are returned (taken) at the hardware level after skipping a specified number of frames.

Syntax

```
DWORD CcmGetTimeDomainTakeFrames(DWORD* frames);
```

Parameters

frames
number of frames that are taken

Notes

Take must be used in conjunction with Skip.

Example

```
DWORD skip, take, samplerate;
float datarate;

CcmGetTimeDomainSkipFrames(&skip);
CcmGetTimeDomainTakeFrames(&take);
CcmGetSampleRate(samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("effective data rate with skip and take = %f\n", datarate);
```

CcmGetUsbConnectionStatus

Description

Returns whether or not the CCM's USB cable has been plugged in to the computer's USB port.

Syntax

```
DWORD CcmGetUsbConnectionStatus(DWORD* isConnected);
```

Parameters

isConnected
on successful return, value is set to 1 if the USB is connected, or 0 if not.

Notes

If the USB cable is plugged in, and this function shows that it cannot connect to USB, this may indicate a bad USB cable, driver issue, or other hardware failure. Try rebooting the PC, using a different cable and a different USB port.

Example

```
DWORD isAvailable;
DWORD isConnected;
DWORD isOpened;

CcmGetUsbConnectionStatus(&isConnected);
if (isConnected == 0)
    printf("USB is not connected\n");
else
{
    CcmGetAvailabilityStatus(&isAvailable);
    if (isAvailable == 0)
        printf("Unable to detect CCM device\n");
    else
    {
        CcmGetDeviceHandleStatus(&isOpened);
        if (isOpened == 0)
           printf("Unable to open device for communication\n");
        else
           printf("CCM device is opened and ready for I/O!\n");
    }
}
```

CcmLoadConfiguration

Description

Load a previously saved configuration from a file.

Syntax

```
DWORD CcmLoadConfiguration(const char* filename);
```

Parameters

filename
null-terminated string of the filename containing the configuration

Notes

The configuration is initially saved using CcmSaveConfiguration.

Example

```
CcmLoadConfiguration("config.cfg");
```

CcmOpen

Description

Open I/O handle for an accessible Command and Control Modules (CCM)

Syntax

```
DWORD CcmOpen(void);
```

Parameters

none

Example

```
if (CcmOpen() != CCM_ERR_NONE)
   printf ("Unable to open devices: err=%d\n", CcmLastError());
```

CcmRead

Description

Read one or more time-domain or frequency-domain frames. Optionally read synchronized images.

Syntax

```
DWORD CcmRead(int*   fdBuffer,
              DWORD  fdElements,
              DWORD* fdFramesRead,
              int*   tdBuffer,
              DWORD  tdElements,
              DWORD* tdFramesRead,
              BYTE*  imageBuffer,
              DWORD  imageBufferBytes,
              DWORD* imageBytesRead,
              DWORD  wait);
```

Parameters

fdBuffer, tdBuffer, imageBuffer
buffers that receive time-domain, frequency-domain, or image data. Can be NULL if either time or frequency domain data is not enabled

tdElements, fdElements
maximum number of 32-bit data elements that can be read into tdBuffer or fdBuffer

tdFramesRead, fdFramesRead
pointer to DWORD that receives the total number of full FD or TD frames read

imageBufferBytes
size of the image buffer in bytes; the maximum image size to be returned

imageBytesRead
pointer to DWORD that receives size in bytes of the returned image

wait
a non-zero value causes the function to wait up to 1 second before returning with less than the requested amount of data. A value of zero causes the function return immediately with any data available

Notes

This function should be used in place of `CcmReadFrames` for new applications.

The *tdTranspose* option available in `CcmReadFrames` has been replaced by the function `CcmSetTimeDomainFrameOrder`.

If synchronized images are returned, the number of frames returned (if available) will always be the value set in the function `CcmSetFramesPerImage`. The number of images returned (if available) will always be one.

Example

```
int tdBuffer[FRAME_SIZE*FRAMES_PER_IMAGE];
int imBuffer[IMAGE_SIZE];
DWORD tdFrames;
DWORD imBytes;

CcmSetFramesPerImage(FRAMES_PER_IMAGE);

CcmRead(NULL, 0, NULL,
        tdBuffer, FRAME_SIZE*FRAMES_PER_IMAGE, &tdFrames,
        imBuffer, IMAGE_SIZE, &imBytes,
        CCM_WAIT);

for (int i=0; i<FRAMES_PER_IMAGE; i++)
   ProcessTimeDomain(tdBuffer, result);

AverageResult(result);
OverlayImage(result, image, imBytes);
```

CcmReadFrames (deprecated)

Description

Read one or more time-domain or frequency-domain frames.

Syntax

```
DWORD CcmReadFrames(int*   tdBuffer,
                    int*   fdBuffer,
                    DWORD  tdElements,
                    DWORD  fdElements,
                    DWORD* tdFramesRead,
                    DWORD* fdFramesRead,
                    DWORD  tdTranspose,
                    DWORD  wait);
```

Parameters

tdBuffer, fdBuffer
time-domain and frequency-domain buffers that receives the data.
Can be NULL if either time or frequency domain data is not enabled

tdElements, fdElements
maximum number of 32-bit data elements that can be read for each buffer

tdFramesRead, fdFramesRead
pointer to DWORD that receives the total number of full FD or TD frames read

tdTranspose
a non-zero value causes time-domain data to be transposed before being returned.

wait
a non-zero value causes the function to wait up to 1 second before returning with less than the requested amount of data. A value of zero causes the function return immediately with any data available

Notes

This function is provided for backwards compatibility. Use CcmRead for new applications.

Example

```
/* create CSV listing of time-domain data */
int tdBuffer[FRAME_SIZE];

CcmReadFrames(buffer, NULL, FRAME_SIZE, 0,
              &tdFrames, &fdFrames, CCM_TRANSPOSE, CCM_WAIT);

for (i=0; i<tdBlockSize; i++)
{
    for (j=0; j<micCount-1; j++)
        printf("%d, ", tdBuffer[i*micCount + j]);

    printf("%d\n", tdBuffer[i*micCount + j]);
}
```

CcmSaveConfiguration

Description

Stores the current CCM configuration to a file

Syntax

```
DWORD CcmSaveConfiguration(const char* filename);
```

Parameters

filename
null-terminated string of the filename containing the configuration

Notes

Reload the saved configuration using CcmLoadConfiguration.

Example

```
/* generate an FFT window and save with the configuration */

CcmWriteFftWindowCoefficients(buffer, blockSize/2, &written);
CcmSelectFftWindow(CCM_WINDOW_CUSTOM);
CcmSaveConfiguration("customWindow.cfg");
```

CcmSetCameraEnabled

Description

Toggle the availablilty of the CCM camera to capture images.

Syntax

```
DWORD CcmSetCameraEnabled(DWORD isEnabled);
```

Parameters

isEnabled

Set to `CCM_ENABLED` to enable the camera, or `CCM_DISABLED` to disable the camera.

Notes

If enabling the camera without first setting the ID using `CcmSetCameraId`, the service will attempt to discover the ID. If unable to find a valid ID, or if the one provided by `CcmSetCameraId` is not a valid camera ID, the function will return with the error `CCM_ERR_CAMERA_NOT_OPENED`.

Disabling the camera is equivalent to calling the function `CcmSetCameraId(CCM_INVALID_CAMERA_ID);`

Example

```
DWORD id;

CcmSetCameraEnabled(CCM_ENABLED);
CcmGetCameraId(&id);

if (id == CCM_INVALID_CAMERA_ID)
    printf("The CCM camera could not be opened\n");
else
    printf("The CCM camera is ready, id=%d\n", id);
```

CcmSetCameraId

Description

Changes the camera used to capture images

Syntax

```
DWORD CcmSetCameraId(DWORD ID);
```

Parameters

ID

The enumerated device ID of the camera to select.

Notes

The CcmService automatically detects the ID of the CCM's onboard webcam when the CCM is plugged into the USB port. A different ID may be selected for an external camera.

Setting ID to `CCM_INVALID_CAMERA_ID` effectively disables camera operations. This is equivalent to calling `CcmSetCameraEnabled(CCM_DISABLED);`

Example

```
DWORD id = CAMERA_ID;

if (CcmSetCameraId(&id) != CCM_ERR_NONE)
    printf("Camera id %d is not available for use.\n", id);
else
    printf("Camera id %d ready!\n", id);
```

CcmSetCameraResolution

Description

Set the current webcam resolution

Syntax

```
DWORD CcmSetCameraId(DWORD resolutionCode);
```

Parameters

resolutionCode

The available codes are found in the file CcmApi.h:

```
CCM_IMAGESIZE_1024x768
CCM_IMAGESIZE_800x600
CCM_IMAGESIZE_640x480
CCM_IMAGESIZE_320x240
```

Example

```
DWORD size

CcmSetCameraResolution(CCM_IMAGESIZE_1024x768);
CcmGetImageSize(&size)
printf("image size for 1024x768 = %d bytes\n", size);

CcmSetCameraResolution(CCM_IMAGESIZE_800x600);
CcmGetImageSize(&size)
printf("image size for 800x600 = %d bytes\n", size);

CcmSetCameraResolution(CCM_IMAGESIZE_640x480);
CcmGetImageSize(&size)
printf("image size for 640x480 = %d bytes\n", size);

CcmSetCameraResolution(CCM_IMAGESIZE_320x240);
CcmGetImageSize(&size)
printf("image size for 320x240 = %d bytes\n", size);
```

CcmSetFramesPerImage

Description

Set the current "frames per image" value used for synchronized camera images.

Syntax

```
DWORD CcmSetFramesPerImage(DWORD fpi);
```

Parameters

fpi
Number of acoustic frames to capture for every image.

Notes

If this value is non-zero, one image is automatically captured for every *fpi* acoustic frames. The application is required to read *fpi* frames and one image at each call to `CcmRead`.

If the value is zero, images are not automatically returned but can be requested asynchronously using the function `CcmCaptureImage`.

Example

```
int tdBuffer[FRAME_SIZE*FRAMES_PER_IMAGE];
int imBuffer[IMAGE_SIZE];
DWORD tdFrames;
DWORD imBytes;

CcmSetFramesPerImage(FRAMES_PER_IMAGE);

CcmRead(NULL, 0, NULL,
        tdBuffer, FRAME_SIZE*FRAMES_PER_IMAGE, &tdFrames,
        imBuffer, IMAGE_SIZE, &imBytes,
        CCM_WAIT);

for (int i=0; i<FRAMES_PER_IMAGE; i++)
   ProcessTimeDomain(tdBuffer, result);

AverageResult(result);
OverlayImage(result, image, imBytes);
```

CcmSetFrequencyDomainBlockSize

Description

Set the block size currently used with frequency-domain data.

Syntax

```
DWORD CcmSetFrequencyDomainBlockSize(DWORD blockSize);
```

Parameters

blockSize

the block size value. The allowed values range from 64 to 2048 in powers of two. The following definitions are available:

```
CCM_BLOCKSIZE_64
CCM_BLOCKSIZE_128
CCM_BLOCKSIZE_256
CCM_BLOCKSIZE_512
CCM_BLOCKSIZE_1024
CCM_BLOCKSIZE_2048
```

Example

```
CcmSetFrequencyDomainBlockSize(CCM_BLOCKSIZE_512);
```

CcmSetFrequencyDomainCustomWindowCoefficients

Description

Write the Fft Window coefficients.

Syntax

```
DWORD CcmSetFrequencyDomainCustomWindowCoefficients
          (const double* buffer,
           DWORD coeffCount,
           DWORD* coeffWritten);
```

Parameters

buffer
double array of window coefficients. Values must be between 0.0 and 1.0

coeffCount
total number of coefficients to write, must be at least the size of the buffer

coeffWritten
pointer to integer that receives the total number of elements written

Notes

The full window must be written in one call, there is no way to write partial windows.

The function `CcmLoadConfiguration` loads a previously saved FFT window if a custom window type was selected.

The window must be symmetrical. Only the first half needs to be written.

Example

```
/* Set triangular window */

double* buffer;
double step;
DWORD windowSize;

windowSize = BLOCKSIZE / 2;
buffer = new double(windowSize);
step = 1.0 / (double)windowSize;

for (i=0; i<BLOCKSIZE/2; i++)
    buffer[i] = (double)i * step;

CcmSetFrequencyDomainCustomWindowCoefficients(buffer, BLOCKSIZE/2, &written);
```

CcmSetFrequencyDomainEnabled

Description

Enable or disable frequency domain output from the CCM

Syntax

```
DWORD CcmSetFrequencyDomainEnabled(DWORD enabled);
```

Parameters

enabled

use CCM_ENABLED to enable, or CCM_DISABLE to disable frequency domain data

Example

```
/* Enable time and frequency domain data */
/* Enable frequency domain frame buffering */
/* Time domain data is the default of continuous frames */

CcmSetFrequencyDomainEnabled(CCM_ENABLED);
CcmSetFrequencyDomainFrameBufferSize(5);
CcmSetTimeDomainEnabled(CCM_ENABLED);
CcmSetTimeDomainFrameBufferSize(5);
```

CcmSetFrequencyDomainFrameBufferSize

Description

Set the number of frames maintained in the frequency domain frame buffer.

Syntax

```
DWORD CcmSetFrequencyDomainFrameBufferSize(DWORD frames);
```

Parameters

frames
number of frames maintained in the frame buffer

Notes

The frame buffer stores only the most recent frames and discards the older ones.

Settnig the frame buffer size to zero disables frame buffering.

Example

```
/* beamforming function requires 5 sequential frames */

CcmSetFrequencyDomainEnabled(CCM_ENABLED);
CcmSetFrequencyDomainFrameBufferSize(5);

/* Now, any successful call to CcmRead will return
   the most recent 5 frequency domain frames */
```

CcmSetFrequencyDomainOverlapEnabled

Description

Set 50% overlap of time-domain data used for the built-in FFT

Syntax

```
DWORD CcmSetFrequencyDomainOverlapEnabled(DWORD enabled);
```

Parameters

enabled
set to CCM_ENABLED to enable or CCM_DISABLED to disable overlap

Notes

Overlap means reusing the last half of the previous time-domain data block in generating a block of frequency domain data. The resulting data rate will be twice the rate if overlap was not enabled.

Example

```
DWORD enabled;

// force overlap to be enabled if not already enabled

CcmGetFrequencyDomainOverlapEnabled(&enabled);
if (enabled == CCM_DISABLED)
    CcmSetFrequencyDomainOverlapEnabled(CCM_ENABLED);
```

CcmSetFrequencyDomainSkipFrames

Description

Set the number of frames that are skipped at the hardware level after taking a specified number of frames.

Syntax

```
DWORD CcmSetFrequencyDomainSkipFrames(DWORD frames);
```

Parameters

frames
number of frames to skip after taking.

Notes

Skip must be used in conjunction with Take.

Example

```
DWORD skip, take, samplerate;
float datarate;

skip = 5;
take = 1;
samplerate = CCM_SAMPLERATE_10000;

CcmSetFrequencyDomainSkipFrames(skip);
CcmSetFrequencyDomainTakeFrames(take);
CcmSetSampleRate(samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("Effective data rate with skip and take = %f Hz\n", datarate);
```

CcmSetFrequencyDomainSourceFile

Description

Sets the name of the binary file to be used as the frequency-domain data source.

Syntax

```
DWORD CcmSetFrequencyDomainSourceFile(const char* filename, DWORD* enabled);
```

Parameters

filename
null-terminated char string containing the name of file to be used

enabled
set to `CCM_ENABLED` or `CCM_DISABLED` indicating whether or not the file has been opened

Notes

The function `CcmStopInput` closes the file and resets the filename to an empty string.

Example

```
DWORD enabled;

CcmSetFrequencyDomainSourceFile("c:\temp\fdTest.bin", &enabled);

if (enabled == CCM_DISABLED)
   printf("Error opening file\n");
else
   printf("File ready");
```

CcmSetFrequencyDomainTakeFrames

Description

Set the number of frames that are returned (taken) at the hardware level after skipping a specified number of frames.

Syntax

```
DWORD CcmSetFrequencyDomainTakeFrames(DWORD frames);
```

Parameters

frames
number of frames to take after skipping

Notes

Take must be used in conjunction with Skip.

Example

```
DWORD skip, take, samplerate;
float datarate;

skip = 5;
take = 1;
samplerate = CCM_SAMPLERATE_10000;

CcmSetFrequencyDomainSkipFrames(skip);
CcmSetFrequencyDomainTakeFrames(take);
CcmSetSampleRate(samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("effective data rate with skip and take = %f\n", datarate);
```

CcmSetFrequencyDomainWindowType

Description

Select one of the pre-defined FFT window types.

Syntax

```
DWORD CcmSetFrequencyDomainWindowType(DWORD window);
```

Parameters

window

FFT window type to apply. The options are:

CCM_FFT_WINDOW_RECTANGULAR
CCM_FFT_WINDOW_HAMMING
CCM_FFT_WINDOW_HANNING
CCM_FFT_WINDOW_CUSTOM

CCM_FFT_WINDOW_CUSTOM requires the application to load a set of FFT window coefficients using the function `CcmSetFrequencyDomainCustomWindowCoefficients`.

Notes

Changing the block size automatically updates any window except Custom windows, which need to be updated using `CcmSetFrequencyDomainCustomWindowCoefficients` to match the new block size.

Example

```
/* generate an FFT window and save with the configuration */

CcmSetFrequencyDomainCustomWindowCoefficients(buffer, blockSize/2, &written);
CcmSetFrequencyDomainWindowType(CCM_WINDOW_CUSTOM);
CcmSaveConfiguration("customWindow.cfg");
```

CcmSetImageBrightness

Description

Set the brightness setting of the camera.

Syntax

```
DWORD CcmSetImageBrightness(DWORD brightness);
```

Parameters

brightness
brightness value. 0 is darkest, 15 is brightest.

Note

The range of the brightness level is determined by the camera. The current array has a range of 0 to 15. The default is 8.

Example

```
value = GetSliderValue();
if (value < 0) value = 0;
if (value > 15) value = 15;

CcmSetImageBrightness(value);
printf("The current brightness level is %d\n", value);
```

CcmSetImageColorMode

Description

Set the color mode of the returned images

Syntax

```
DWORD CcmSetImageColorMode(DWORD code);
```

Parameters

code

one of the possible color mode codes found in CcmApi.h:

CCM_IMAGECOLOR_RGB24:	24-bit color (default)
CCM_IMAGECOLOR_BW8:	8-bit black and white

Note

Reducing the color depth to 8-bit black and white decreases the file size of uncompressed images by a factor of three, but may not significantly decrease the size of compressed images.

Example

```
CcmSetImageColorMode(CCM_IMAGECOLOR_BW8);
printf("Now returning black and white images!\n");
```

CcmSetImageFormat

Description

Set the format of the images returned by the API

Syntax

```
DWORD CcmSetImageFormat(DWORD fmt);
```

Parameters

fmt

one of the possible format codes found in CcmApi.h:

```
CCM_IMAGEFORMAT_JPG    - default
CCM_IMAGEFORMAT_BMP
CCM_IMAGEFORMAT_PNG
CCM_IMAGEFORMAT_TIF
CCM_IMAGEFORMAT_MAT    - returns data part of OpenCv matrix
```

Note

With the exception of `CCM_IMAGEFORMAT_MAT`, each returned image is complete - that is, the buffer can be stored to disk and opened with an appropriate viewer. No extra processing of the image data is required.

Example

```
DWORD size;

CcmSetImageFormat(CCM_IMAGEFORMAT_JPG);
CcmGetImageSize(&size)
printf("max image size for JPG = %d bytes\n", size);

CcmSetImageFormat(CCM_IMAGEFORMAT_BMP);
CcmGetImageSize(&size)
printf("image size for BMP = %d bytes\n", size);

CcmSetImageFormat(CCM_IMAGEFORMAT_TIF);
CcmGetImageSize(&size)
printf("image size for TIF = %d bytes\n", size);

CcmSetImageFormat(CCM_IMAGEFORMAT_PNG);
CcmGetImageSize(&size)
printf("max image size for PNG = %d bytes\n", size);

CcmSetImageFormat(CCM_IMAGEFORMAT_MAT);
CcmGetImageSize(&size)
printf("opencv matrix size = %d bytes\n", size);
```

CcmSetJpgCompression

Description

Set the JPG image compression level

Syntax

```
DWORD CcmSetJpgCompression(DWORD compression);
```

Parameters

compression

compression level:

0 = maximum compression (smallest file size, poor quality)
100 = minimum compression (largest file size, higher quality)

The default JPG compression level is 95

Example

```
DWORD size;

CcmSetImageType(CCM_IMAGETYPE_JPG);

CcmSetJpgCompression(100);
CcmGetImageSize(&size)
printf("image size for low compression (100) = %d bytes\n", size);

CcmSetJpgCompression(95);
CcmGetImageSize(&size)
printf("image size for default compression (95) = %d bytes\n", size);

CcmSetJpgCompression(70);
CcmGetImageSize(&size)
printf("image size for medium compression (70) = %d bytes\n", size);

CcmSetJpgCompression(30);
CcmGetImageSize(&size)
printf("image size for high compression (30) = %d bytes\n", size);
```

CcmSetLogfileRoot

Description

Sets the base folder used for server logging

Syntax

```
DWORD CcmSetLogfileRoot(const char* path);
```

Parameters

path
null-teriminated string of the new base folder

Notes

If not set, or if set to an empty string, the default folder is C:\Data\Signal Interface Group.

Example

```
CcmSetLogfileRoot("C:\\TestData");
```

CcmSetPngCompression

Description

Set the PNG image compression level

Syntax

```
DWORD CcmSetPngCompression(DWORD compression);
```

Parameters

compression

the current compression level:

0 = minimum compression effort (larger file size, shorter compression time)
9 = maximum compression effort (smaller file size, longer compression time)

The default PNG compression level is 3

Notes

PNG is a lossless compression, so images have the same quality regardless of the compression level selected. The only effect is on the file size. Typically, the difference in file size between moderate compression effort and and high compression effort is minimal.

Example

```
DWORD size;

CcmSetImageType(CCM_IMAGETYPE_PNG);

CcmSetPngCompression(0);
CcmGetImageSize(&size)
printf("image size for low compression (0) = %d bytes\n", size);

CcmSetPngCompression(3);
CcmGetImageSize(&size)
printf("image size for default compression (3) = %d bytes\n", size);

CcmSetPngCompression(9);
CcmGetImageSize(&size)
printf("image size for high compression (9) = %d bytes\n", size);
```

CcmSetPowerState

Description

Turn CCM power on or off

Syntax

```
DWORD CcmSetPowerState(DWORD power);
```

Parameters

power
set to CCM_POWER_ON or CCM_POWER_OFF

Notes

If power is already on, turning the power on again cycles power providing a full reset.

Example

```
CcmSetPowerState(CCM_POWER_ON);
```

CcmSetSampleRate

Description

Set the sample rate in Hz

Syntax

```
DWORD CcmSetSampleRate(DWORD rate);
```

Parameters

rate

the new sample rate in Hz. The valid sample rates are:

```
CCM_SAMPLERATE_10000
CCM_SAMPLERATE_12500
CCM_SAMPLERATE_20000
CCM_SAMPLERATE_25000
CCM_SAMPLERATE_40000
CCM_SAMPLERATE_50000
```

Example

```
if (CcmSetSampleRate(CCM_SAMPLERATE_50000) != CCM_ERR_NONE)
    printf("Unable to read sample rate: err = %d\n", CcmGetLastError());
else
    printf("Sample rate = %d Hz\n", CCM_SAMPLERATE_50000);
```

CcmSetTimeDomainBlockSize

Description

Set the block size currently used with time-domain data.

Syntax

```
DWORD CcmSetTimeDomainBlockSize(DWORD blockSize);
```

Parameters

blockSize

the block size value. The allowed values range from 64 to 2048 in powers of two. The following definitions are available:

```
CCM_BLOCKSIZE_64
CCM_BLOCKSIZE_128
CCM_BLOCKSIZE_256
CCM_BLOCKSIZE_512
CCM_BLOCKSIZE_1024
CCM_BLOCKSIZE_2048
```

Example

```
DWORD blocksize = CCM_BLOCKSIZE_128;

CcmSetTimeDomainBlockSize(blocksize);
printf("Time domain block size is %d\n", blocksize);
```

CcmSetTimeDomainEnabled

Description

Enable or disable time domain output from the CCM

Syntax

```
DWORD CcmSetTimeDomainEnabled(DWORD enabled);
```

Parameters

enabled

use CCM_ENABLED to enable, or CCM_DISABLE to disable time domain data

Example

```
/* Enable time and frequency domain data */
/* Enable frequency domain frame buffering */
/* Time domain data is the default of continuous frames */

CcmSetFrequencyDomainEnabled(CCM_ENABLED);
CcmSetFrequencyDomainFrameBufferSize(5);
CcmSetTimeDomainEnabled(CCM_ENABLED);
CcmSetTimeDomainFrameBufferSize(5);
```

CcmSetTimeDomainFrameBufferSize

Description

Set the number of frames maintained in the time domain frame buffer.

Syntax

```
DWORD CcmSetTimeDomainFrameBufferSize(DWORD frames);
```

Parameters

frames
number of frames maintained in the frame buffer

Notes

The frame buffer stores only the most recent frames and discards the older ones.

Setting frames to zero disables frame buffering. Setting to non-zero enables.

Example

```
/* beamforming function requires 5 sequential TD frames */

CcmSetTimeDomainEnabled(CCM_ENABLED);
CcmSetTimeDomainFrameBufferSize(5);

/* Now, any successful call to CcmRead will return
   the most recent 5 time domain frames */
```

CcmSetTimeDomainFrameOrder

Description

Set the orientation of returned time domain frames to block order (default) or scan order (transpose).

Syntax

```
DWORD CcmSetTimeDomainFrameOrder(DWORD order);
```

Parameters

order
frame order code:

`CCM_FRAMEORDER_BLOCK` (default)
`CCM_FRAMEORDER_SCAN` (transpose)

Notes

`CCM_FRAMEORDER_SCAN` combined with CcmRead has the same functionality as using `CCM_TRANSPOSE` with the deprecated CcmReadFrames function.

Example

```
CcmSetTimeDomainFrameOrder(CCM_FRAMEORDER_SCAN);
```

CcmSetTimeDomainSkipFrames

Description

Set the number of frames that are skipped at the hardware level after taking a specified number of frames.

Syntax

```
DWORD CcmSetTimeDomainSkipFrames(DWORD frames);
```

Parameters

frames
number of frames to skip after taking.

Notes

Skip must be used in conjunction with Take.

Example

```
DWORD skip, take, samplerate;
float datarate;

skip = 5;
take = 1;
samplerate = CCM_SAMPLERATE_10000;

CcmSetTimeDomainSkipFrames(skip);
CcmSetTimeDomainTakeFrames(take);
CcmSetSampleRate(samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("Effective data rate with skip and take = %f Hz\n", datarate);
```

CcmSetTimeDomainSourceFile

Description

Sets the name of the binary file to be used as the time-domain data source.

Syntax

```
DWORD CcmSetTimeDomainSourceFile(const char* filename, DWORD* enabled);
```

Parameters

filename
null-terminated char string containing the name of file to be used

enabled
set to `CCM_ENABLED` or `CCM_DISABLED` indicating whether or not the file has been opened

Notes

The function `CcmStopInput` closes the file and resets the filename to an empty string.

Example

```
DWORD enabled;

CcmSetTimeDomainSourceFile("c:\temp\tdTest.bin", &enabled);

if (enabled == CCM_DISABLED)
   printf("Error opening file\n");
else
   printf("File ready");
```

CcmSetTimeDomainTakeFrames

Description

Set the number of frames that are returned (taken) at the hardware level after skipping a specified number of frames.

Syntax

```
DWORD CcmSetTimeDomainTakeFrames(DWORD frames);
```

Parameters

frames
number of frames to take after skipping

Notes

Take must be used in conjunction with Skip.

Example

```
DWORD skip, take, samplerate;
float datarate;

skip = 5;
take = 1;
samplerate = CCM_SAMPLERATE_10000;

CcmSetTimeDomainSkipFrames(skip);
CcmSetTimeDomainTakeFrames(take);
CcmSetSampleRate(samplerate);

datarate = (float)samplerate * (float)take / (float)(take+skip);
printf("effective data rate with skip and take = %f\n", datarate);
```

CcmStartFrequencyDomainLogging

Description

Start frequency domain service logging.

Syntax

```
DWORD CcmStartFrequencyDomainLogging(void);
```

Parameters

none

Notes

The path for the logged data must be set using the function CcmSetLogfilePath before calling this function or data will be written to C:\Data\Signal Interface Group.

If called before StartInput, data will be logged immediately after input is started.

Calling StopInput automatically stops logging.

Example

```
CcmStartFrequencyDomainLogging();
```

CcmStartInput

Description

Start the input sampling clock.

Syntax

```
DWORD CcmStartInput(void);
```

Parameters

none

Example

```
CcmStartInput();
printf("Input sampling started.\n");
```

CcmStartLogging

Description

Simultaneously start both time-domain and frequency-domain service logging.

Syntax

```
DWORD CcmStartLogging(void);
```

Parameters

none

Note s

Service logging can be started with any combination of `CcmStartLogging`, `CcmStartTimeDomainLogging` and `CcmStartFrequencyDomainLogging`. For example, calling `CcmStartLoggingEnabled`() followed by `CcmStopTimeDomainLogging`() will keep frequency-domain logging active.

The path for the logged data must be set using the function CcmSetLogfilePath before calling this function or data will be written to C:\Data\Signal Interface Group.

If called before StartInput, data will be logged immediately after input is started.

Calling StopInput automatically stops logging.

Example

```
CcmStartLogging();
```

CcmStartTimeDomainLogging

Description

Start time domain service logging.

Syntax

```
DWORD CcmStartTimeDomainLogging(void);
```

Parameters

none

Notes

The path for the logged data must be set using the function CcmSetLogfilePath before calling this function or data will be written to C:\Data\Signal Interface Group.

If called before StartInput, data will be logged immediately after input is started.

Calling StopInput automatically stops logging.

Example

```
CcmStartInput();
CcmStartTimeDomainLogging();
/* Do something */
CcmStopTimeDomainLoggingEnabled();
CcmStopInput();
```

CcmStopFrequencyDomainLogging

Description

Stop frequency domain service logging.

Syntax

```
DWORD CcmStopFrequencyDomainLogging(void);
```

Parameters

none

Notes

Calling StopInput automatically stops logging, and calling this function is not necessary.

Example

```
CcmStopFrequencyDomainLogging();
```

CcmStopInput

Description

Stop the input sampling clock.

Syntax

```
DWORD CcmStopInput(void);
```

Parameters

none

Example

```
CcmStopInput();
printf("Input sampling stopped.\n");
```

CcmStopLogging

Description

Simultaneously stop both time-domain and frequency-domain service logging.

Syntax

```
DWORD CcmStopLogging(void);
```

Parameters

none

Note s

Calling StopInput automatically stops logging, and calling this function is not necessary.

Example

```
CcmStopLogging();
```

CcmStopTimeDomainLogging

Description

Stop time domain service logging.

Syntax

```
DWORD CcmStopTimeDomainLogging(void);
```

Parameters

none

Notes

Calling StopInput automatically stops logging, and calling this function is not necessary.

Example

```
CcmStopTimeDomainLogging();
```

Appendix B: Error Codes

Appendix B contains an alphabetical listing of the CcmAccess DLL errors, each with its message text and further a description of the error.

Most CcmApi DLL function calls return an error code, or CCM_ERR_NONE if no error was encountered. The error code can be retrieved using the function CcmGetLastError, and the error text can be retrieved using the function CcmGetLastErrorString.

The error codes are located in the file CcmErrors.h.

CCM_ERR_ALREADY_OPENED:
"Device is already open”
Attepted to call CcmOpen when already opened.

CCM_ERR_AUTHENTICATION:
"Authentication failed”
The control module firmware failed its validity test. Please contact Signal Interface Group for more help in troubleshooting this error.

CCM_ERR_BAD_DLL_MESSAGE:
"The service received an invalid, or out-of-sequence message from the DLL”
Each message sent from the DLL to the service contains a message code and message sequence. In this case, the sequence was out-of-order, or the code was not recognized by the service.

CCM_ERR_BAD_PATH:
"Unable to create directory. Check Event Viewer for target path"
The DLL was not able to create the directory specified for service logging.

CCM_ERR_BAD_SERVICE:
"Unable to connect to CcmService”
The DLL was not able to connect to the service. Check that the service is not stopped.

CCM_ERR_BAD_SERVICE_MESSAGE:
"The CcmApi DLL received an invalid, or out-of-sequence message from CcmService”
Each message sent from the DLL to the service expects the service to send a response message. In this case, the message code or sequence did not match the expected response.

CCM_ERR_BAD_SOURCE_FILE:
"Unable to open data source file”
The file specified by CcmSetTimeDomainSourceFile or CcmSetFrequencyDomainSourceFile could not be opened for reading. Make sure the path is correct and that the file is not open in another application.

CCM_ERR_BUFFER_TOO_SMALL:
"The buffer supplied is too small for the data to be read”
The data or image buffer supplied to CcmRead is too small to complete the operation.

CCM_ERR_CAMERA_ASYNC_NOT_ALLOWED:
"Cannot capture images asynchronously with FramesPerImage > 0”
CcmCaptureImage was called after CcmSetFramesPerImage had set the framesPerImage count greater than

zero. With framesPerImage > 0, only synchronized images may be read using CcmRead.

CCM_ERR_CAMERA_IMAGE_TOO_LARGE:
"Image too large for pipe - see event viewer”
Images are being backed-up in the service. Verify that the application is reading images after setting CcmSetFramesPerImage to greater than zero. If CcmRead is returning valid images, this may be an internal error.

CCM_ERR_CAMERA_NOT_OPENED:
"Cannot find selected video input device”
This usually means that the camera has not been enabled. It could also mena that the service was unable to detect the CCM camera, or the specified alternate camera ID is invalid.

CCM_ERR_CCM_NOT_AVAIL:
"Cannot establish communication with CCM”
An attempt was made to turn the power on to a CCM, but the service could not detect the board.

CCM_ERR_CONFIG_INCOMPATIBLE:
"Configuration does not match the detected hardware"
The number of devices in the configuration must match the physical number of devices in the system.

CCM_ERR_CONFIG_MISMATCH:
"Service and DLL configurations are incompatible"
The version of the service which is currently installed cannot be used with the version of the DLL which is currently installed. Reinstalling the CcmAccess software may be required.

CCM_ERR_DEVICE_IN_USE:
"Device is being used by another application"
An attempt was made to open a handle to a device that has already been opened by another application.

CCM_ERR_DISCONNECTED:
"Device is not connected"
The USB cable has been unplugged.

CCM_ERR_DLL_MESSAGE_NOT_SENT:
"The CcmApi DLL was unable to send a message to CcmService”
The DLL was not able to write the complete message. Check that the service is not stopped.

CCM_ERR_FLASH_READ_TIMEOUT:
"Unable to read all requested data from flash"
Only some, or none, of the configuration data was successfully read from the flash memory. Try disconnecting and reconnecting the device's USB cable and try again.

CCM_ERR_FLUSH_TIMEOUT:
"Timeout while flushing"
All data could not be flushed in the alloted time. Make sure I/O is stopped, and/or repeat the call.

CCM_ERR_INIT_REG_READ:
"Unable to read registers during initialization"
Certain registers are read during initialization that are required to properly control the hardware. This error indicates that these registers could not be read. It may be necessary to cycle power to reset hardware.

CCM_ERR_INVALID_COEFFICIENT:

"Invalid calibration coefficient. Value must be between 0.0 and <2.0"
Bad calibration coefficient encountered.

CCM_ERR_INVALID_DEVICE:
"Invalid device number"
USB driver error. The CCM hardware may need to be repowered.

CCM_ERR_INVALID_HANDLE:
"Invalid handle"
USB driver error. The CCM hardware may need to be repowered.

CCM_ERR_INVALID_NULL_POINTER:
"Application has passed an invalid NULL pointer to a CcmApi function"
NULL pointer passed to a function where not allowed.

CCM_ERR_INVALID_PARAMETER:
"HW parameter error"
Low level USB driver error. This may require reinstalling the USB driver.

CCM_ERR_INVALID_PIPE:
"Internal operation encountered an invalid pipe code"
Unexpected routing data was received by the CCM device. Check that the device is supported by the DLL.

CCM_ERR_INVALID_SAMPLE_RATE:
"The selected sample rate is not supported by this array"
Different sets of sample rates are supported depending on the model. The available sample rates can be obtained using the function CcmGetSampleRateList.

CCM_ERR_IO:
"HW I/O Error"
USB driver error. Check the USB connection.

CCM_ERR_IO_ACTIVE:
"Operation not possible while I/O is active"
Configuration, and some other API functions, are not available while I/O is active.

CCM_ERR_LOGGING_ACTIVE
"Operation not possible while logging is active"
Certain functions, such as `CcmSetLogfilePath`, cannot be called while logging is enabled.

CCM_ERR_MISSING_CONFIG
"Unable to load configuration from flash"
Uninitialized or possibly damaged device.

CCM_ERR_NONE:
"No error"
The last CcmAccess function call completed successfully.

CCM_ERR_NOT_CALIBRATED:
"Calibrated results are requested but coefficients are not available"
Calibration may need to be restored, if possible, or disabled.

CCM_ERR_NOT_FOUND:

"Hardware not detected"
No CCM hardware is present.

CCM_ERR_NOT_OPENED:
"Device has not been opened"
Attempt to perform an operation on a CCM which does not exist or has not yet been opened with CcmOpen

CCM_ERR_PARAM_OUT_OF_RANGE:
"Parameter out of range"
A parameter to an API function is out of range.

CCM_ERR_CCM_POWER_OFF:
"Device power has been turned off"
An attempt was made to access the CCM after the power had been turned off.

CCM_ERR_RAW_DATA_ERROR:
"An error in the raw data stream has prevented the data from being interpreted correctly”
This may be a result of not properly flushing old data, an interruption of the USB communication, or the hardware may be incompatable with the CcmService software.

CCM_ERR_READ_TIMEOUT:
"Timeout while reading"
Unable read the requested amount of data within the alloted time

CCM_ERR_REGISTER_READ_FAIL:
"Unable to read register value"
An updated register value could not be obtained within the allotted time.

CCM_ERR_RESOURCES:
"HW resource error"
USB driver has trouble allocating resources.

CCM_ERR_SERVICE_MESSAGE_NOT_SENT:
"CcmService was unable to send a message to the CcmApi DLL”
The service was not able to write the complete message.

CCM_ERR_SERVICE_UNAVAILABLE:
"CcmService is stopped or is unavailable”
Check the Control Panel services program to verify that CcmService is installed and running. If it is not installed, please refer to the CcmAccess installation instructions.

CCM_ERR_SERVICE_UNKNOWN:
"CcmService has encountered an unknown error”
An error has forced the service to reset the CCM hardware. Applications must close and reopen device handles to continue. Continued errors may require reboot. This is a "catch-all" error that should never be encountered.

CCM_ERR_UNABLE_TO_OPEN_CONFIG:
"Unable to open file for configuration”
The DLL was not able to access the requested configuration file.

CCM_ERR_UNABLE_TO_OPEN_LOGFILE:
"Unable to open logfile(s)”

The service was not able to create the logfiles.

CCM_ERR_UNABLE_TO_WRITE_CONFIG:
"Unable to write configuration XML to service”
The DLL was not able to send the XML configuration to CcmService.

CCM_ERR_UNKNOWN:
"Unknown error"
"Catch-All" for all API functions. This error occurs if a non-CcmAccess or non-Windows error is thrown and no description is possible.

CCM_ERR_UNKNOWN_SYSTEM:
"Unhandled Windows exception"
"Catch-All" for all Windows functions if no message is available

CCM_ERR_USB_NOT_CONNECTED:
"USB cable is not connected"
Unable to detect any supported USB device

CCM_ERR_USB_NOT_OPENED:
"USB device handle has not been opened"
USB devices are detected, but have not been properly opened for access by CcmService.

Appendix C. Configuration XML Example

The following XML example shows a typical configuration. Descriptions and comments shown in italics are not part of the actual XML file, and some repetitive fields are condensed by "..."

The first several lines of the configuration XML lists hardware, firmware and software properties

```
<CcmConfiguration>
  <Company>Signal Interface Group</Company>
  <VersionInfo>
    <ConfigVersion>
      <Major>1</Major>
      <Minor>0</Minor>
      <Revision>0</Revision>
    </ConfigVersion>
    <ServiceVersion>
      <Major>1</Major>
      <Minor>0</Minor>
      <Revision>0</Revision>
    </ServiceVersion>
    <DllVersion>
      <Major>1</Major>
      <Minor>0</Minor>
      <Revision>0</Revision>
    </DllVersion>
  </VersionInfo>
  <Hardware>
    <Arrays>
      <NumArrays>1</NumArrays>
      <Array0>
        <Controller>
          <Model>CCM 2000</Model>
          <Desc>Controller Module</Desc>
          <Sn>99999</Sn>
          <FirmwareVersion>
            <Major>1</Major>
            <Minor>0</Minor>
            <Revision>0</Revision>
          </FirmwareVersion>
          <SrRangeEnum>0</SrRangeEnum>
        </Controller>
        <Settings>
```

The configuration settings controlled by the application start here. Most fields are self-explanatory.

```
          <SampleRate>10000</SampleRate>
          <Calibration>
            <Enabled>1</Enabled>
            <Coefficients>
              <NumCoeff>40</NumCoeff>
              <Coeff0>1.000000</Coeff0>
              <Coeff1>1.000000</Coeff1>
                 ...
```

```
        <Coeff39>1.000000</Coeff39>
      </Coefficients>
    </Calibration>
    <TimeDomain>
      <Enabled>1</Enabled>
      <Logging>1</Logging>
      <BlockSize>64</BlockSize>
      <Skip>0</Skip>
      <Take>1</Take>
      <FrameBuffering>
        <Enabled>0</Enabled>
        <NumFrames>5</NumFrames>
      </FrameBuffering>
    </TimeDomain>
    <FrequencyDomain>
      <Enabled>1</Enabled>
      <Logging>0</Logging>
      <Overlap>0</Overlap>
      <BlockSize>64</BlockSize>
      <Skip>0</Skip>
      <Take>1</Take>
      <FrameBuffering>
        <Enabled>0</Enabled>
        <NumFrames>5</NumFrames>
      </FrameBuffering>
```

<Window> shows the FFT windowing vector settings. The <WindowType> value corresponds to the "CCM_FFT_WINDOW_*" constants found in CcmApi.h. Type 2 shown here means a Hamming window has been selected. If a custom window is entered (WindowType = 1), the list of coefficients will be listed below the window type.

```
      <Window>
        <WindowType>2</WindowType>
      </Window>
    </FrequencyDomain>

    <CameraSettings>
      <Id>0</Id>
      <Enabled>1</Enabled>
      <CalEnabled>1</CalEnabled>
      <Resolution>
        <Code>102</Code>
        <Width>640</Width>
        <Height>480</Height>
      </Resolution>
      <Format>
        <Code>1</Code>
        <Jpg>95</Jpg>
        <Png>3</Png>
      </Format>
      <ColorMode>1</ColorMode>
      <Brightness>8</Brightness>
      <FramesPerImage>0</FramesPerImage>
    </CameraSettings>

  </Settings>
  <MicrophonePcb>
```

```
          <NumPcb>1</NumPcb>
          <Pcb0>
            <Microphones>
```

The microphone coordinates are installed when the board is first initialized and cannot be changed.

```
              <NumMics>40</NumMics>
              <Coordinates>
                <Units>mm</Units>
                <Mic0>
                  <X>  55.0</X>
                  <Y>-3163.8</Y>
                </Mic0>
                  ...
                <Mic39>
                  <X>  55.0</X>
                  <Y>-3163.8</Y>
                </Mic39>
              </Coordinates>
            </Microphones>
          </Pcb0>
        </MicrophonePcb>
      </Array0
    </Arrays>
  </Hardware>
</CcmConfiguration>
```

Appendix D. Handling Precautions

Static control is required for handling all electronic equipment. The Microphone Array and Command and Control Module (CCM) PCBs are sensitive to static discharge because of the analog and digital components. To protect the boards, observe the following precautions:

- Do not touch any exposed connectors.
- Provide a chassis ground when possible, especially in environments where there is a high probability of static discharge.
- When transporting, make sure the array is wrapped in the conductive plastic bag that it was shipped with. If this is not available, shield the board by wrapping it completely in aluminum foil. Do not transport, ship, or store a board without suitable conductive shielding.

Appendix E. Contact Information

For more information, contact:

Signal Interface Group
2265 116th Ave NE
Bellevue, WA 98004
Telephone: 425-467-7146
Email: sales@signalinterface.com