C API Reference

Shadow Version 4.0

www.motionshadow.com

Contents

1 Introduction

The Motion C Application Progamming Interface (API) is a set of plain C functions that provides real-time access to the output of the Motion Service. The C API is available as a standalone dynamic library suitable for use in applications such as LabVIEW and Matlab.

The C API depends on the open source Motion Software Development Kit (SDK) for the underlying communication and formatting systems. Refer to the **SDK Reference** document for more information.

Just like the SDK and the desktop application, the C API accesses sensor data streams from the Motion Service software. All device configuration and management is handled through the desktop application or scripting commands.

For convenience, the C API will attempt to scan for and start any available devices when a client opens a connection.

2 Reference

The C API uses only built in types and has no external dependencies. All functions are thread safe. The library runs its own sampling loops independent of the client application. The client may choose any polling rate and even use the non-blocking read functionality to simplify integration.

For sample size, channel order, and unit information refer to the **Real-time Data Streams** section of the **SDK Reference** document.

2.1 Functions

`int mncapi_open(mncapi_stream_t type)`

Summary	Open a connection to a real-time data stream. Select `MNCAPI_PREVIEW`, `MNCAPI_SENSOR`, or `MNCAPI_RAW` stream type at connect time.
Postcondition	`handle > 0`
Parameter	`type` stream type identifier
Return	positive integer valued handle for a connection resource, 0 value indicates error

```
#include <MotionCAPI.h>
#include <stdio.h>

int main(int argc, char * argv[])
{
  int handle = mncapi_open(MNCAPI_SENSOR);
  if (handle > 0)
  {
    float data[9];
    if (mncapi_sample(handle, data, 9) > 0)
    {
      /* Have a sample of sensor data, print it out. */
      printf("a = [%f, %f, %f] g\n", data[0], data[1], data[2]);
      printf("m = [%f, %f, %f] uT\n", data[3], data[4], data[5]);
      printf("g = [%f, %f, %f] deg/sec\n",
             data[6], data[7], data[8]);
      printf("\n");
    }

    mncapi_close(handle);
  }

  return 0;
}
```

Example 1: API example usage. Connect, read calibrated sensor data, and then close.

```
int mncapi_open_host(mncapi_stream_t type,
     const char * host,
     int port)
```

Summary Same as `mncapi_open` with the addition of host and port fields.
Postcondition `handle > 0`
Parameter `type` stream type identifier
`host` ip address string
`port` integer port number, set to -1 to use default port
Return positive integer valued handle for a connection resource, 0 value indicates error

```
int mncapi_open_configurable(const char * xml,
     const char * host,
     int port)
```

Summary Same as `mncapi_open_host` with the addition of the XML definition field. Open a connection to the Configurable data service. Choose which channels to read by an XML definition string.
Postcondition `handle > 0`
Parameter `xml` configurable service defintion in XML format
`host` ip address string
`port` integer port number, set to -1 to use default port
Return positive integer valued handle for a connection resource, 0 value indicates error

```
void mncapi_close(int handle)
```

Summary Close an open connection to a real-time data stream.
Parameter `handle` integer valued handle for a connection resource, set to `0` to close all existing connections

```
int mncapi_set_blocking(int handle, int second)
```

Summary Set blocking time out or behavior for calls to `mncapi_sample` and other reading calls. The default behavior is a one second time out, `MNCAPI_DEFAULT`. To disable time out, use `MNCAPI_FOREVER`. To disable blocking and return the most recent sample immediately, use `MNCAPI_NOBLOCK`.
Parameter `handle` integer valued handle for a connection resource
`second` integer valued time out value
Return `MNCAPI_SUCCESS` if blocking parameter set, otherwise `MNCAPI_FAILURE`

int mncapi_set_buffered(int handle, int num)

Summary Set the buffer size for the connection sampling loop. The connection will store up to `num` samples of data between calls to `mncapi_sample`. Set to `0` for unlimited buffer size. Set to `1` for non-buffered stream. Set to `2` or more to set a limit on the buffer size. If the limit is exceeded the connection will fail.

Parameter `handle` integer valued handle for a connection resource
`num` integer valued number or samples

Return `MNCAPI_SUCCESS` if buffer parameter set, otherwise `MNCAPI_FAILURE`

int mncapi_sample_ready(int handle)

Summary Query function to read the number of samples available in the buffer. Non-blocking call only counts samples already available. If there is a sample available, a subsequent call to `mncapi_sample` will never block.

Parameter `handle` integer valued handle for a connection resource

Return integer valued number of samples available, otherwise `MNCAPI_FAILURE`

int mncapi_sample(int handle,
float * data,
int data_size)

Summary Read data from a real-time data stream. Requires an existing connection handle. Note that this is intended for use with the preview and sensor data streams. Use `mncapi_sample_int16` for raw data streams.

Precondition `data` must have at least `data_size` elements
`data_size` must be large enough to hold at least one complete sample

Parameter `handle` integer valued handle for a connection resource
`data` array to write sample data to
`data_size` size of data array

Return number of devices successfully read and copied to the `data` buffer, otherwise `MNCAPI_FAILURE`

int mncapi_sample_int16(int handle,
short * data,
int data_size)

Summary Same as `mncapi_sample` except that this is intended for use with a raw data stream.

```
int mncapi_get_preview(float * data)
```

Summary	Simplified interface using built in resource handle. Similar to `mncapi_sample(MNCAPI_PREVIEW, data, 14)`.
Precondition	`data` must have at least `14` elements

```
int mncapi_get_sensor(float * data)
```

Summary	Simplified interface using built in resource handle. Similar to `mncapi_sample(MNCAPI_SENSOR, data, 9)`.
Precondition	`data` must have at least `9` elements

```
int mncapi_get_raw(short * data)
```

Summary	Simplified interface using built in resource handle. Similar to `mncapi_sample_int16(MNCAPI_RAW, data, 9)`.
Precondition	`data` must have at least `9` elements

```
int mncapi_lua_chunk(const char * input,
     char * output,
     int output_size)
```

Summary	Send a Lua string to the local console service. Return the printed results.
Precondition	`output` must have at least `output_size` elements
Postcondition	`output` contains the null terminated character string
Parameter	`input` Lua formatted scripting chunk, null terminated character string `output` printed results from the execution of the entire Lua chunk `output_size` number of bytes in the output buffer

```
int mncapi_lua_chunk_host(const char * host,
     const char * input,
     char * output,
     int output_size)
```

Summary	Same as `mncapi_lua_chunk` except allows for remote console service connnection.
Parameter	`host` ip address string

2.2 Calling Conventions

The C API functions are declared with the `__cdecl` calling convention. The header file reflects this convention.

To maximize compatibility, the library also includes `__stdcall` versions of all functions. The `__stdcall` function names are postfixed with the name `_stdcall`. For example, call `mncapi_open` through the function named `mncapi_open_stdcall`.