C Client Library

V3DSClientLib provides a strict C function interface. Any language that supports C, such C, Objective-C, Java, Matlab and Python, can use V3DSClientLib to access real-time data from Visual3DServer. The most important functions are: ''%%v3ds_init()%%'' and ''%%v3ds_kill()%%'' ''%%v3ds_connect()%%'' and ''%%v3ds_disconnect()%%'' ''%%v3ds_setSystems()%%'', ''%%v3ds_setPipeline()%%'' and ''%%v3ds_setLinkModel()%%'' ''%%v3ds_installCallback()%%'' for clients that are pushed data ''%%v3ds_getValue()%%'' or ''%%v3ds_getResults()%%'' for clients that poll for data The remaining functions are mostly for examining and adjusting the internal state of V3DSClientLib. C and C++ clients have the option of polling for data or being pushed data through an installed C callback function. Matlab and Python clients must poll for data currently. [[#Example_Client_Code Example client code]] is provided in ANSI C, C, Matlab and Python to help you get started using V3DSClientLib.

V3DSClientLib

---

bool v3ds_init(...)

- **Arguments:** `const int knTcpTimeout` - the TCP timeout in milliseconds (ms).`const int knTcpBufferSize` - the TCP buffer size in bytes (B).`const int knUdpBufferSize` - the UDP buffer size in bytes (B).`const int knSleep` - the thread sleep in milliseconds (ms).`const char* pchPriority` - pointer to a character array containing the thread priority.
- **Return:** `bool` - returns `true` if the library is successfully initialized, otherwise returns `false`.
- **Description:** Initializes the library by allocating the TCP and UDP buffers.

Recommend default values are: - TCP timeout: 5000 ms - TCP buffer size: 16383 B - UDP buffer size: 32767 B - thread sleep: 1 ms - thread priority: “normal” Thread priority can be one of “Critical”, “Highest”, “High”, “Normal”, “Low”, “Lowest”, “Idle”. Thread priority is case insensitive.

---

bool v3ds_kill()

- **Arguments:** none
- **Return:** `bool` - returns `true` if the library was successfully killed, otherwise returns `false`.
- **Description:** This function will:1) disconnect the client from Visual3DServer (if connected),2) stop the data acquisition thread,3) de-allocate the TCP and UDP buffers.

---

int v3ds_getTcpTimeout()

- **Arguments:** none
- **Return:** `int` - returns the TCP timeout in milliseconds (ms).

- **Description:** The TCP timeout is how long the client waits for a response from Visual3DServer. The TCP timeout is set when the client calls `v3ds_init()`.

---

int v3ds_getTcpBufferSize()

- **Arguments:** none
- **Return:** `int` - returns the TCP buffer size in bytes (B).
- **Description:** The TCP buffer size is the amount of memory allocated for network messages sent between the client and Visual3DServer. The TCP buffer size is set when the client calls `v3ds_init()`.

Importnat: the TCP buffer must be large enough to hold the longest possible message. 16384 bytes is the recommended minimum size.

---

int v3ds_getUdpBufferSize()

- **Arguments:** none
- **Return:** `int` - returns the UDP buffer size in bytes (B).
- **Description:** The UDP buffer size is the amount of memory allocated for network data sent from Visual3DServer to the client. The UDP buffer size is set when the client calls `v3ds_init()`.

Important: the UDP buffer must be large enough to hold the longest possible data results. 32676 bytes is the recommended minimum size. Note that the data results length is typically proportional to the length of the pipeline script set in the client.

---

const char* v3ds_getPriority()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to a character array containing the thread priority.
- **Description:** Thread priority can be one of “Critical”, “Highest”, “High”, “Normal”, “Low”, “Lowest”, “Idle”. Thread priority is case insensitive. The thread priority is set when the client calls `v3ds_init()`.

---

int v3ds_getSleep()

- **Arguments:** none
- **Return:** `int` - returns the thread sleep in milliseconds (ms).
- **Description:** The thread sleep is the amount of time the library waits (sleeps) at the end of

the data acquisition loop for the UDP socket.

A small sleep value makes the data acquisition loop run faster, minimizing the UDP packet size and the possibly of missing frames from Visaul3DServer. However, less CPU time will be available to other programs. A larger sleep value makes the data acquisition loop run slower, increasing the UDP packet size and the possibly of missing frames from Visual3DServer. However, more CPU time will be available to other programs. In general, a lower sleep value is appropriate, especially if Visual3DServer or the 3rd party mocap software is running on a different machine. Thread sleep is set when the client calls `v3ds_init()`.

---

bool v3ds_connect(...)

- **Arguments:** `char* pchIP` - Visaul3DServer IP address.`int nTcpPort` - Visual3DServer TCP broadcast port number.`char* pchName` - Client application name.`char* pchVersion` - Client application version.`char* pchDate` - Client application release date.`char* pchEnvrionment` - Client application environment.`char* pchProtocol` - Client application protocol.
- **Return:** `bool` - returns `true` if the client connects to Visual3DServer, otherwise returns `false`.
- **Description:** The IP address refers to the machine that is running Visual3DServer.If the client and Visual3DServer are running on the same machine, then the IP address can be “localhost” (case insensitive) or 127.0.0.1.If Visual3DServer is running on a different machine anywhere on the internet, then the IP address is that of the host machine.Use a shell command like 'ipconfig' to determine a machine's IP address and 'ping' to confirm network traffic is possible.

The port number for Visual3DServer is 12099. Here are examples for the descriptive client strings: Client name: “MyClientApp”–any client name is allowed. Client version: “1.0.0”–any version number format is allowed. Client date: “2014-Aug-25”–any date format is allowed. Client environment: “Windows8Pro”–any environment description is allowed. Client protocol: “Biofeedback”–any protocol description is allowed.

---

bool v3ds_isConnected()

- **Arguments:** none
- **Return:** `bool` - returns `true` if the client is connected to Visual3DServer, otherwise returns `false`.
- **Description:** Indicates if a connection is present.

---

bool v3ds_disconnect()

- **Arguments:** none
- **Return:** `bool` - returns `true` a the client was disconnected, otherwise returns `false`.
- **Description:** Disconnects the client from Visaul3DServer, if connected.

---

const char* v3ds_getClientName()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the client name character array.
- **Description:** The library owns the memory for the client name. The client should never delete or alter this memory and should treat it as read only.

---

const char* v3ds_getClientVersion()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the client version character array.
- **Description:** The library owns the memory for the client version. The client should never delete or alter this memory and should treat it as read only.

---

const char* v3ds_getClientDate()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the client date character array.
- **Description:** The library owns the memory for the client date. The client should never delete or alter this memory and should treat it as read only.

---

const char* v3ds_getClientEnvironment()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the client environment character array.
- **Description:** The library owns the memory for the client environment. The client should never delete or alter this memory and should treat it as read only.

---

const char* v3ds_getClientProtocol()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the client protocol character array.
- **Description:** The library owns the memory for the client protocol. The client should never delete or alter this memory and should treat it as read only.

---

const char* v3ds_getTCPIPAddress()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the TCP/IP address.
- **Description:** The library owns the memory for the TCP/IP address. The client should never delete or alter this memory and should treat it as read only.

---

int v3ds_getUDPPort()

- **Arguments:** none
- **Return:** `int` - returns the client specific UDP broadcast port number.
- **Description:** Visual3DServer assigns a unique UDP broadcast port number to each client upon connection.

---

bool v3ds_setSystems(char*)

- **Arguments:** none
- **Return:** `char*` - comma separated, prioritized, list of systems the client wants to select.
- **Description:** Selecting a system means that, on a frame-by-frame basis, Visual3DServer will apply the client's pipeline script to that system and return results accordingly.

A client can select more than one system and also set the priority order of those systems.

---

bool v3ds_isSystemsSet()

- **Arguments:** none
- **Return:** `bool` - returns `true` if the client has selected one or more systems, otherwise returns `false`.
- **Description:** Indicates if any systems are currently selected.

---

const char* v3ds_getSystems()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the comma separated list of selected systems.
- **Description:** The library owns the memory for the selected systems list. The client should never delete or alter this memory and should treat it as read only.

---

bool v3ds_setPipeline(char*)

- **Arguments:** `char* pchPipeline` - the pipeline to set.
- **Return:** `bool` - returns `true` if the pipeline script was set, otherwise returns `false`.
- **Description:** Indicates if the pipeline script was set.

---

v3ds_isPipelineSet

- **Arguments:** none
- **Return:** `bool` - returns `true` if the client has set a pipeline script, otherwise returns `false`.
- **Description:** Indicates if a pipeline script is currently set.

---

const char* v3ds_getPipeline()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the current pipeline script.
- **Description:** The library owns the memory for the pipeline script. The client should never delete or alter this memory and should treat it as read only.

---

bool v3ds_setLinkModel(char*)

- **Arguments:** `char* pchLinkModel` - the link model to set.
- **Return:** `bool` - returns `true` if the link model was set, otherwise returns `false`.
- **Description:** Indicates if the link model was set.

---

bool v3ds_isLinkModelSet()

- **Arguments:** none
- **Return:** `bool` - returns `true` if the client has set a link model, otherwise returns `false`.
- **Description:** Indicates if a link model is currently set.

---

const char* v3ds_getLinkModel()

- **Arguments:** none
- **Return:** `const char*` - returns a pointer to the current link model.
- **Description:** The library owns the memory for the current link model. The client should never delete or alter this memory and should treat it as read only.

bool v3ds_parseXML(char*, void*)

- **Arguments:** `char* pXML` - pointer to a character array containing XML from Visual3DServer.`void* pdeqRes` - pointer to a void memory block. See description for details.
- **Return:** `bool` - returns `true` if new data is available, otherwise returns `false`.
- **Description:** This function parses the XML data coming from Visual3DServer and stores the results in memory.

If the pdeqRes argument is NOT 0, then the results are stored in the pdeqRes argument and are accessible Cclients immediately. If the pdeqRes argument is 0, then the results are stored internally and are accessed by the client using the ''%%v3ds_getResult..()%%'' functions. The client must ensure the mutex is locked before calling any ''%%v3ds_getResult..()%%'' functions. See [[#bool_v3ds_lockMutex() `v3ds_lockMutex()`]]. —- === bool v3ds_lockMutex() === * **Arguments:** none * **Return:** `bool` - returns `true` if there is new data available, otherwise returns `false`. * **Description:** Locks the mutex so the library does overwrite memory with new results UNTIL the client calls `v3ds_unlockMutex()`. This function is used when a client is polling for data. While the mutex is locked, the client can call any of the `v3ds_getResult..()` functions. The client should spend as little time as possible between calls to `v3ds_lockMutex()` and `v3ds_unlockMutex()`. The mutex will ONLY be locked if there is new data available when this function is called. Otherwise the mutex will NOT be locked when this function returns. —- === void v3ds_unlockMutex() === * **Arguments:** none * **Return:** `void` - does not return a value. * **Description:** Unlocks the mutex so V3DSClientLib can access the data memory to store new results. —- === int v3ds_getResultsCount() === * **Arguments:** none * **Return:** `int` - returns the number of results. * **Description:** This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === const char* v3ds_getResultString(const int) === * **Arguments:** `const int knResIdx` - result index. `pchDest`. * **Return:** `const char*` - returns a pointer to a character array containing the result. * **Description:** This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === const char* v3ds_getResultType(const int) === * **Arguments:** `const int knResIdx` - constant integer containing result index. * **Return:** `const char*` - returns a pointer to a character array containing the result TYPE. * **Description:** The memory for the result TYPE character array is owned by the library. The client should never delete this memory. This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === const char* v3ds_getResultFolder(const int) === * **Arguments:** `const int knResIdx` - constant integer containing result index. * **Return:** `const char*` - returns a pointer to a character array containing the result FOLDER. * **Description:** The memory for the result FOLDER character array is owned by the library. The client should never delete this memory. This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === const char* v3ds_getResultName(const int) === * **Arguments:** `const int knResIdx` - constant integer containing result index. * **Return:** `const char*` - returns a pointer to a character array containing the result NAME. * **Description:** The memory for the result NAME character array is owned by the library. The client should never delete this memory. This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === int v3ds_getResultFramesCount(const int) === * **Arguments:** `const int knResIdx` - constant integer containing the result index. * **Return:** `int` - returns the number of frames for the given result. * **Description:** This function might alter the status string. The client must ensure the mutex is locked before calling this function.

See ''%%v3ds_lockMutex()%%''. —- === int v3ds_getResultFrameValue(const int, const int) === * **Arguments:** `const int knResIdx` - result index.`const int knFrameIdx` - subframe index. * **Return:** `int` - returns the frame index for the given result and subframe. * **Description:** The frame index is usually determined by the hardware and is unique for each frame. This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === int v3ds_getResultFrameValuesCount(const int, const int) === * **Arguments:** `const int knResIdx` - result index.`const int knFrameIdx` - subframe index. * **Return:** `int` - returns the number of values or components per subframe for the result. * **Description:** This function returns the number of components per subframe for the result. This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === float v3ds_getResultValue(const int, const int, const int) === * **Arguments:** `const int knResIdx` - result index.`const int knFrameIdx` - subframe index.`const int knCompIdx` - component index. * **Return:** `float` - returns result value. * **Description:** All indices are automatically checked internally to ensure they do not exceed array bounds. This function might alter the status string. The client must ensure the mutex is locked before calling this function. See ''%%v3ds_lockMutex()%%''. —- === void* v3ds_getResultsObject() === * **Arguments:** none * **Return:** `void*` - returns a void pointer to the memory holding the results. * **Description:** C clients that poll for data can use this function to gain direct memory access to the latest results. Cclients should cast the ''%%void*%%'' to ''%%std::deque<CResType>*%%''. See the C++ Example Client Code. The client must ensure the mutex is locked before calling this function. See [[#bool_v3ds_lockMutex() `v3ds_lockMutex()`]]. . —- === void v3ds_installCallback(*pCallback, void*) === * **Arguments:** `bool (*pCallback)(void*, char*, int)` - pointer to callback function. Passing in `0` removes an existing callback.`void* pUserData` - pointer to user data. * **Return:** `void` - does not return a value. See `v3ds_isCallbackInstalled()` instead. * **Description:** Installs the callback function provided by the client. This function is used to 'push' data to the client so that the client does not have to poll for data. Using a 'push' data callback reduces the chances of missing frames over polling for data. —- === bool v3ds_isCallbackInstalled() === * **Arguments:** none * **Return:** `bool` - returns true if a data callback is installed, otherwise returns false. * **Description:** Indicates if the client has installed a data callback function. This function is used to 'push' data to the client so that the client does not have to poll for data. Using a 'push' data callback reduces the chances of missing frames over polling for data.Also see `v3ds_installCallback()`. —- === const char* v3ds_getStatus() === * **Arguments:** none * **Return:** `const char*` - returns “OK” or a string describing the internal state of the library. An empty string is never returned. * **Description:** The status string is updated internally as various functions are called in the library. The memory for the status string is owned by the library. The client should never delete this memory. —- ==== Example Client Code ==== Example client code is provided in ANSI C, C++, Matlab and Python to help you get started using V3DSClientLib. Note that the examples generally contain less error checking code and more global variables than a typical real-world program. This was done so that the important details of using V3DSClientLib would stand out. Ideally your clients would be more fully and robustly designed. —- === ANSI C Example Client Code ===

```
/*****************************************************************************
****
| ANSI C ConsoleClient Example
| Copyright (C) 2014 C-Motion, Inc.
|*****************************************************************************
***/
```

```
/* Copy V3DSClientLib.dll to client application folder. */
/* Copy Qt5Core.dll and Qt5Network.dll to client application folder. */
/* Copy Qt platform folder to client application folder. */

/* Link to V3DSClientLib.lib to client application. */

/* Include V3DSClientLib header */
#include "../V3DSClientLib/V3DSClientLib.h"

/* Global Variables */
static int      g_nCount        = 0;        // How many results did we
count?
static bool     bPollForData    = false;    // false means use callback
instead.

/* Global functions */
static bool     dataCallback(void* pUserData, char* pXML, int nMsgCount);
static bool     printData();

/*************************************************************************
****
| main
|*************************************************************************
***/

int main(int argc, char* argv[])
{
    /* Initialize V3DSClientLib. */
    bool bContinue = v3ds_init(5000, 16384, 32768, 1, "normal");

    if(bContinue)
    {
        /* Connect to Visual3DServer. */
        bContinue = v3ds_connect("127.0.0.1", 12099, "ConsoleClient", "1.0",
            "Today", "env", "Biofeedback");
    }

    /* Initialize count variables for counting results. */
    g_nCount = 0;
    int nMaxCount = 5;

    if(bContinue && !bPollForData)
    {
        /* Install callback if bPollForData is false. */
        v3ds_installCallback(dataCallback, 0);
    }

    if(bContinue)
    {
        /* Set selected system to C3D File. */
        bContinue = v3ds_setSystems("C3D");
```

```
    }

    if(bContinue)
    {
        /* Set pipeline script to look for LTOE marker. */
        char* pchPipelineScript =
            "Multiply_Signals_By_Constant"
            "/SIGNAL_TYPES=TARGET"
            "/SIGNAL_FOLDER=ORIGINAL"
            "/SIGNAL_NAMES=LTOE"
            "/RESULT_TYPES=TARGET"
            "/RESULT_FOLDER=ORIGINAL"
            "/RESULT_NAMES=LTOE"
            "/RESULT_SUFFIX=_RT"
            "/SIGNAL_COMPONENTS=0"
            "/CONSTANT=1;";
        bContinue = v3ds_setPipeline(pchPipelineScript);
    }

    /* Look for nMaxCount results in a fast loop. */
    while(g_nCount < nMaxCount)
    {
        /* Check if we are polling instead of the callback method. */
        if(bPollForData)
        {
            /* We are polling.  Print any new reuslts. */
            if(v3ds_lockMutex())
            {
                printData();
                v3ds_unlockMutex();
            }
        }
    }

    /* Tear down V3DSClientLib before quitting application. */
    v3ds_kill();

    return 0;
}

/*************************************************************************
****
| dataCallback -- invoked when bPollForData is false
|*************************************************************************
***/

static bool dataCallback(void* pUserData, char* pXML, int nMsgCount)
{
    bool bHaveData = false;
```

```
    // mutex is automatically locked while callback is invoked.

    // 0 for 2nd arg means we are not a C++ client, so we need to use
    // v3ds_get..() functions to get data in printData().
    if(v3ds_parseXML(pXML, 0))
    {
        bHaveData = printData();
    }

    // mutex is automatically unlocked when callback returns.

    return bHaveData;
}

/*************************************************************************
****
| printData -- called either when polling or using the callback method
|*************************************************************************
***/

static bool printData()
{
    bool bHaveData = false;

    int nNumRes = v3ds_getResultsCount();
    if(0 < nNumRes)
    {
        const bool kbUseStringFunc = true;

        for(int r = 0; r < nNumRes; r++)
        {
            if(kbUseStringFunc)
            {
                const int knBufSize = 4096; // make it something big
                char chBuf[knBufSize];
                if(0 < v3ds_getResultString(r, chBuf, knBufSize))
                {
                    printf("%s", chBuf);
                }
                else
                {
                    printf("No data found for result %d.\n", r);
                }
            }
            else    // ask for each piece of data separately.
            {
                printf("Result name: %s\n", v3ds_getResultName(r));
                int nSubFrames = v3ds_getResultFramesCount(r);
                if(0 <  nSubFrames)
                {
                    int nFrameVal = v3ds_getResultFrameValue(r, 0);
```

// Copy V3DSClientLib.dll to client application folder. // Copy Qt5Core.dll and Qt5Network.dll to client application folder. // Copy Qt platform folder to client application folder.
// Link to V3DSClientLib.lib to client application.

```
                    printf("Subframes: %d, subframe %d value: %d\n",
                        nSubframes, 1, nFrameVal);

                    int nNumCompPerFrame =
v3ds_getResultFrameValuesCount(r);

                    printf("Components: %d\n", nNumCompPerFrame);
                    for(int c = 0; c < nNumCompPerFrame; c++)
                    {
                        printf("Component %d value: %f\n", c+1,
                            v3ds_getResultValue(r, 0, c));
                    }
                }
            }
        }

        g_nCount++; /* main() is examining this variable in a while loop. */

        bHaveData = true;
    }
    return bHaveData;
}
/*****************************************************************************
***/
// === C Example Client Code ===
int nMaxCount = 5;
```

The C example is very similar to the C example. The notable difference is the optional use of a ''%%std::deque<CResType>%%'' to hold the results coming from Visaul3DServer. C++ clients can directly access this deque of results instead of calling various ''%%v3ds_get..()%%'' functions. When using a callback function, C++ clients can pass a deque of results to ''%%v3ds_parseXML()%%'' to be populated. When polling, C++ clients can call ''%%v3ds_getResults()%%'' to get a deque of results. The return from ''%%getResults()%%'' must be cast from ''%%void*%%'' to ''%%std::deque<CResType>*%%''. <code>

```
/*****************************************************************************
```

C ConsoleClient Example application

```
    if(bContinue)
    {
        // Set selected system to C3D File.
        bContinue = v3ds_setSystems("C3D");
    }

    if(bContinue)
    {
        // Set pipeline script to look for LTOE marker.
        char* pchPipelineScript =
            "Multiply_Signals_By_Constant"
            "/SIGNAL_TYPES=TARGET"
            "/SIGNAL_FOLDER=ORIGINAL"
```

```
        "/SIGNAL_NAMES=LTOE"
        "/RESULT_TYPES=TARGET"
        "/RESULT_FOLDER=ORIGINAL"
        "/RESULT_NAMES=LTOE"
        "/RESULT_SUFFIX=_RT"
        "/SIGNAL_COMPONENTS=0"
        "/CONSTANT=1;";
    bContinue = v3ds_setPipeline(pchPipelineScript);
}
```

```
std::deque<CResType>* pdeqRes = 0;
```

```
// Look for nMaxCount results in a fast loop.
while(g_nCount < nMaxCount)
{
    // Check if we are polling instead of the callback method.
    if(bPollForData)
    {
        // We are polling.  Print any new reuslts.
        if(v3ds_lockMutex())
        {
            void* p = v3ds_getResultsObject();
            if(0 != p)
            {
                // Magic cast to access the CResType memory objects.
                pdeqRes = reinterpret_cast<std::deque<CResType>*>(p);
                printData(*pdeqRes);
```

```
                g_nCount++;
            }
            v3ds_unlockMutex();
        }
    }
}
```

```
// Tear down V3DSClientLib before quitting application.
v3ds_kill();
```

```
return 0;
```

} static bool dataCallback(void* pUserData, char* pXML, int nMsgCount) {

```
std::cout << "dataCallback()" << std::endl;
```

```
bool bHaveData = false;
```

```
if(v3ds_parseXML(pXML, &g_deqRes))
{
    bHaveData = printData(g_deqRes);
```

```
      g_nCount++; // main() is examining this variable in a while loop.
  }
```

```
  return bHaveData;
```

} static bool printData(const std::deque<CResType>& deqRes) {

```
  std::cout << "printData()" << std::endl;
```

```
  bool bHaveData = false;
```

```
  const size_t knNumRes = deqRes.size();
  if(0 < knNumRes)
  {
      const bool kbUseStringFunc = true;
```

```
      std::cout << "Results: " << knNumRes << std::endl;
      for(size_t r = 0; r < knNumRes; r++)
      {
          if(kbUseStringFunc)
          {
              std::cout << deqRes[r];
          }
          else    // ask for each piece of data separately.
          {
              std::cout <<
                  "Name: " << deqRes[r].m_strName.c_str() <<
                  ", Folder: " << deqRes[r].m_strFolder.c_str() <<
                  ", Type: " << deqRes[r].m_strType.c_str() << std::endl;
```

```
              const size_t knSubframes = deqRes[r].m_deqFrame.size();
              std::cout << "Subframes: " << knSubframes << std::endl;
              for(size_t f = 0; f <  knSubframes; f++)
              {
                  std::cout << "FrameId: " <<
deqRes[r].m_deqFrame[f].m_nFrameId << std::endl;
                  std::cout << "Timestamp: " <<
deqRes[r].m_deqFrame[f].m_fTime << std::endl;
                  const size_t knNumComp =
deqRes[r].m_deqFrame[f].m_deqfVal.size();
                  std::cout << "Components: " << knNumComp << std::endl;
                  if(0 < knNumComp)
                  {
                      std::cout << "(";
                      for(size_t c = 0; c < knNumComp; c++)
                      {
                          std::cout << deqRes[r].m_deqFrame[f].m_deqfVal[c];
                          if(c < knNumComp-1)
                          {
```

```
                            std::cout << ",";
                        }
                    }
                    std::cout << ")" << std::endl;
                }
            }
        }
    }
```

```
    bHaveData = true;
  }
```

```
  return bHaveData;
```

} </code> —- === Matlab Example Client Code === * **Requirements** Matlab R2013b * **If coding TCP socket and XML parsing (not necessary using V3DSClientLib), then** Instrument Control Toolbox for TCP and UDP support)http://www.mathworks.com/

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   client.m
%
%   Demonstrates how to use the V3DSClientLib matlab package to communicate
%   with the Visual3DServer application.
%
%   This matlab file works with data/run.c3d streaming in Visual3DServer.
%
%   Copyright (C) 2013-2014 C-Motion, Inc.  All rights reserved.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load Visual3DServer client library, if necessary.
addpath(strcat(getenv('CM_V3DS_DIR'), '\SDK'));
addpath(strcat(getenv('CM_V3DS_DIR'), '\SDK\ClientCode\Matlab'));
addpath(strcat(getenv('CM_V3DS_DIR'), '\SDK\Release'));

if not(libisloaded('V3DSClientLib'))
    loadlibrary('V3DSClientLib', 'V3DSClientLib.h');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if libisloaded('V3DSClientLib')
    disp('Initializing V3DSClientLib...');
    % Initial object to represent Visual3DServer (V3DS).
    bContinue = calllib('V3DSClientLib', 'v3ds_init', 5000, 4096, 8192, ...
        5, 'normal');
    disp(calllib('V3DSClientLib', 'v3ds_getStatus'));
```

```
    if bContinue
        disp('Connecting to V3DSClientLib...');
        % Connect to Visual3DServer with IP address, etc.
        bContinue = calllib('V3DSClientLib', 'v3ds_connect', ...
            '127.0.0.1', 12099, 'MatlabClient', '1.0', '01-Jan-2014', ...
            'env', 'Biofeedback');
        disp(calllib('V3DSClientLib', 'v3ds_getStatus'));
    else
        bContinue = calllib('V3DSClientLib', 'v3ds_kill');
        unloadlibrary('V3DSClientLib');
        return
    end

    if bContinue
        disp('Setting systems in V3DSClientLib...');
        % Tell V3DS that we want data from the C3D File data source.
        bContinue = calllib('V3DSClientLib', 'v3ds_setSystems', 'C3D');
        disp(calllib('V3DSClientLib', 'v3ds_getStatus'));
    else
        bContinue = calllib('V3DSClientLib', 'v3ds_kill');
        unloadlibrary('V3DSClientLib');
        return
    end

    if bContinue
        disp('Setting pipeline script in V3DSClientLib...');
        % Tell V3DS what pipeline script to execute on each frame of data.
        % Pipeline script can be inline, as done here, or loaded from a
file.
        pipelineScript = ['Multiply_Signals_By_Constant' ...
            '/SIGNAL_TYPES=TARGET' ...
            '/SIGNAL_FOLDER=ORIGINAL' ...
            '/SIGNAL_NAMES=LTOE' ...
            '/RESULT_FOLDER=ORIGINAL' ...
            '/RESULT_TYPE=TARGET' ...
            '/RESULT_NAME=LTOE' ...
            '/RESULT_SUFFIX=_RT' ...
            '/CONSTANT=1.0;' ...
            'Multiply_Signals_By_Constant' ...
            '/SIGNAL_TYPES=ANALOG' ...
            '/SIGNAL_FOLDER=ORIGINAL' ...
            '/SIGNAL_NAMES=FZ1' ...
            '/RESULT_FOLDER=ORIGINAL' ...
            '/RESULT_TYPE=ANALOG' ...
            '/RESULT_NAME=FZ1' ...
            '/RESULT_SUFFIX=_RT' ...
            '/CONSTANT=1.0;'];
        bContinue = calllib('V3DSClientLib', 'v3ds_setPipeline', ...
            pipelineScript(1:end));
```

```
        disp(calllib('V3DSClientLib', 'v3ds_getStatus'));
    else
        bContinue = calllib('V3DSClientLib', 'v3ds_kill');
        unloadlibrary('V3DSClientLib');
        return
    end

    if bContinue
        disp('Looking for 5 frames of data...');
        startTime = tic;     % Note the start time.

        % Get pipeline command results now over and over.
        nNumFramesRecv = 0;
        nTotalNumFrames = 5;
        while nNumFramesRecv < nTotalNumFrames
            calllib('V3DSClientLib', 'v3ds_lockMutex');
            bClientResults = calllib('V3DSClientLib', 'v3ds_hasNewResults');
            if bClientResults
                nNumFramesRecv = nNumFramesRecv + 1;
                nNumRes = calllib('V3DSClientLib', 'v3ds_getNumResults');
                fprintf('Found %d result(s)\n', nNumRes);
                for r = 0:nNumRes-1
                    strName = calllib('V3DSClientLib', 'v3ds_getName', r);
                    fprintf('Result %d: %s\n', r+1, strName);
                    nSubFrames = calllib('V3DSClientLib',
'v3ds_getNumFrames', r);
                    if 0 < nSubFrames
                        nFrameVal = calllib('V3DSClientLib',
'v3ds_getFrameValue', r, 0);
                        fprintf('Subframes: %d, subframe %d value: %d\n',
...
                            nSubFrames,  0, nFrameVal);
                        nNumComps = calllib('V3DSClientLib',
'v3ds_getNumCompsPerFrame', r);
                        fprintf('Components: %d\n', nNumComps);
                        for c = 0:nNumComps-1
                            fVal = calllib('V3DSClientLib', 'v3ds_getValue',
r, 0, c);
                            fprintf('Component %d value: %f\n', c+1, fVal);
                        end
                    end
                end
            end
            calllib('V3DSClientLib', 'v3ds_unlockMutex');
        end

        % Note how long it took to request frames
        fprintf('Elapsed time: %0.6f\n', toc(startTime));

        % Disconnect from Visual3DServer.
        calllib('V3DSClientLib', 'v3ds_disconnect');
```

```
    else
        disp(calllib('V3DSClientLib', 'v3ds_getStatus'));
        bContinue = calllib('V3DSClientLib', 'v3ds_kill');
        unloadlibrary('V3DSClientLib');
        return
    end

    % Tear down object that represents Visual3DServer (V3DS).
    bContinue = calllib('V3DSClientLib', 'v3ds_kill');

    if libisloaded('V3DSClientLib')
        unloadlibrary('V3DSClientLib');
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

—- === Python Example Client Code === * **Requirements** Python 2.7.x (http://www.python.org/). * **Recommend** Microsoft Visual Studio (2010 edition)Python Tools for Visual Studio (http://pytools.codeplex.com/) v2.0 for VS 2010PyOpenGL-3.0.2 (OpenGL library support) (http://pyopengl.sourceforge.net/)PyQt4 (Qt library support) (http://www.riverbankcomputing.com/)

```
############################################################################
####
#
#   client.py
#
#   Demonstrates how to use the V3DSClientLib python package to communicate
#   with the Visual3DServer application.
#
#   Copyright (C) 2013-2014 C-Motion, Inc.  All rights reserved.
#
############################################################################
####

############################################################################
####
# Imports
############################################################################
####

# Standard package imports
from ctypes import *
import time

v3dsClientLibPath = 'C:/Program Files (x86)/C-
Motion/Visual3DServer/SDK/Release/V3DSClientLib.dll'
v3dsClientLib = cdll.LoadLibrary(v3dsClientLibPath)
```

```
v3dsClientLib.v3ds_init.restype = c_bool
v3dsClientLib.v3ds_kill.restype = c_bool

v3dsClientLib.v3ds_getTcpTimeout.restype = c_int
v3dsClientLib.v3ds_getTcpBufferSize.restype = c_int
v3dsClientLib.v3ds_getUdpBufferSize.restype = c_int

v3dsClientLib.v3ds_getPriority.restype = c_char_p
v3dsClientLib.v3ds_getSleep.restype = c_int

v3dsClientLib.v3ds_connect.restype = c_bool
v3dsClientLib.v3ds_isConnected.restype = c_bool
v3dsClientLib.v3ds_disconnect.restype = c_bool

v3dsClientLib.v3ds_getClientName.restype = c_char_p
v3dsClientLib.v3ds_getClientVersion.restype = c_char_p
v3dsClientLib.v3ds_getClientDate.restype = c_char_p
v3dsClientLib.v3ds_getClientEnvironment.restype = c_char_p
v3dsClientLib.v3ds_getClientProtocol.restype = c_char_p

v3dsClientLib.v3ds_setSystems.restype = c_bool
v3dsClientLib.v3ds_isSystemsSet.restype = c_bool
v3dsClientLib.v3ds_getSystems.restype = c_char_p

v3dsClientLib.v3ds_setPipeline.restype = c_bool
v3dsClientLib.v3ds_isPipelineSet.restype = c_bool
v3dsClientLib.v3ds_getPipeline.restype = c_char_p

v3dsClientLib.v3ds_setLinkModel.restype = c_bool
v3dsClientLib.v3ds_isLinkModelSet.restype = c_bool
v3dsClientLib.v3ds_getLinkModel.restype = c_char_p

v3dsClientLib.v3ds_hasNewResults.restype = c_bool
v3dsClientLib.v3ds_getNumResults.restype = c_int
v3dsClientLib.v3ds_getResultString.restype = c_char_p
v3dsClientLib.v3ds_getName.restype = c_char_p
v3dsClientLib.v3ds_getNumFrames.restype = c_int
v3dsClientLib.v3ds_getFrameValue.restype = c_int
v3dsClientLib.v3ds_getNumCompsPerFrame.restype = c_int
v3dsClientLib.v3ds_getValue.restype = c_float

v3dsClientLib.v3ds_getStatus.restype = c_char_p

################################################################################
####
# Script
################################################################################
####

# Create object to represent Visual3DServer (V3DS).
print('Initializing V3DSClientLib...')
```

```
bContinue = v3dsClientLib.v3ds_init(5000, 4096, 8192, 5, b"normal")
print('Status: ' + v3dsClientLib.v3ds_getStatus())

# Connect to Visual3DServer with IP address, etc.
if bContinue:
    print('Connecting to V3DSClientLib...')
    bContinue = v3dsClientLib.v3ds_connect(b"127.0.0.1", 12099,
b"Python27Client", \
        b"1.0", b'20-Dec-2013', b'env', b'Biofeedback')
    print('Status: ' + v3dsClientLib.v3ds_getStatus())
else:
    exit()

if bContinue:
    print('Getting systems in V3DSClientLib...')
    # Ask V3DS the systems.
    strSystems = v3dsClientLib.v3ds_getSystems()
    print('Systems: ' + strSystems)
    print('Status: ' + v3dsClientLib.v3ds_getStatus())
else:
    exit()

if bContinue:
    print('Setting systems in V3DSClientLib...')
    # Tell V3DS the systems, in priority order, in which to receive data.
    bContinue = v3dsClientLib.v3ds_setSystems(b'C3D')
    print('Status: ' + v3dsClientLib.v3ds_getStatus())
else:
    exit()

if bContinue:
    print('Setting pipeline script in V3DSClientLib...')
    # Tell V3DS what pipeline script to execute on each frame of data.
    # Pipeline script can be inline, as done here, or loaded from a file.
    pipelineScript = \
        b'Multiply_Signals_By_Constant' \
        b'/SIGNAL_TYPES=TARGET' \
        b'/SIGNAL_FOLDER=ORIGINAL' \
        b'/SIGNAL_NAMES=LTOE' \
        b'/RESULT_FOLDER=ORIGINAL' \
        b'/RESULT_TYPE=TARGET' \
        b'/RESULT_NAME=LTOE' \
        b'/RESULT_SUFFIX=_RT' \
        b'/CONSTANT=1.0;' \
        b'Multiply_Signals_By_Constant' \
        b'/SIGNAL_TYPES=ANALOG' \
        b'/SIGNAL_FOLDER=ORIGINAL' \
        b'/SIGNAL_NAMES=FZ1' \
        b'/RESULT_FOLDER=ORIGINAL' \
```

```
        b'/RESULT_TYPE=ANALOG' \
        b'/RESULT_NAME=FZ1' \
        b'/RESULT_SUFFIX=_RT' \
        b'/CONSTANT=1.0;'
    bContinue = v3dsClientLib.v3ds_setPipeline(pipelineScript)
    print('Status: ' + v3dsClientLib.v3ds_getStatus())
else:
    exit()

if bContinue:
    print('Looking for 5 frames of data...')

    startTime = time.clock()   # Note the start time.

    nNumFramesRecv = 0
    nTotalNumFrames = 5
    while nNumFramesRecv < nTotalNumFrames:
        v3dsClientLib.v3ds_lockMutex()
        clientResults = v3dsClientLib.v3ds_hasNewResults()
        if clientResults:
            nNumFramesRecv += 1
            nNumRes = v3dsClientLib.v3ds_getNumResults()
            print('Found ' + str(nNumRes) + ' result(s)')
            for r in range(0, nNumRes):
                strName = str(v3dsClientLib.v3ds_getName(r))
                print('Result ' + str(r+1) + ': ' + strName)
                nSubFrames = v3dsClientLib.v3ds_getNumFrames(r)
                if 0 < nSubFrames:
                    nFrameVal = v3dsClientLib.v3ds_getFrameValue(r, 0)
                    print('Subframes: ' + str(nSubFrames) + \
                        ', subframe 1 value: ' + str(nFrameVal))
                    nNumComps = v3dsClientLib.v3ds_getNumCompsPerFrame(r)
                    print('Components: ' + str(nNumComps))
                    for c in range(0, nNumComps):
                        fVal = v3dsClientLib.v3ds_getValue(r, 0, c)
                        print('Component ' + str(c+1) + ' value: ' +
str(fVal))


        v3dsClientLib.v3ds_unlockMutex()

    # Note how long it took to request frames
    print('Elapsed time: ' + str(round(time.clock() - startTime, 6)))

    # Disconnect from Visual3DServer.
    v3dsClientLib.v3ds_disconnect()
else:
    print(v3dsClientLib.v3ds_getStatus())
    exit()

print('Status: ' + v3dsClientLib.v3ds_getStatus())
```

```
# Tear down object that represents Visual3DServer (V3DS).
bContinue = v3dsClientLib.v3ds_kill()

##########################################################################
####
```

—-