

## DSRemote library

```
hllDll = WinDLL ("C:\\DXEProjects\\Tools\\DSRemoteConnect\\DSRemoteConnect\\Debug\\DSRemoteConnect64.dll")
dsconInstance = HANDLE()
doErrCheck(hllDll.dsconCreateInstance(pointer(dsconInstance), 1), "dsconCreateInstance")
charArray = create_string_buffer("10.2.120.255:8999:8001".encode())
doErrCheck(hllDll.dsconConnect(dsconInstance, charArray), "dsconConnect")
numChannels = c_size_t()
doErrCheck(hllDll.dsconGetChannelCount(dsconInstance, byref(numChannels)), "dsconGetChannelCount")
print(numChannels.value)

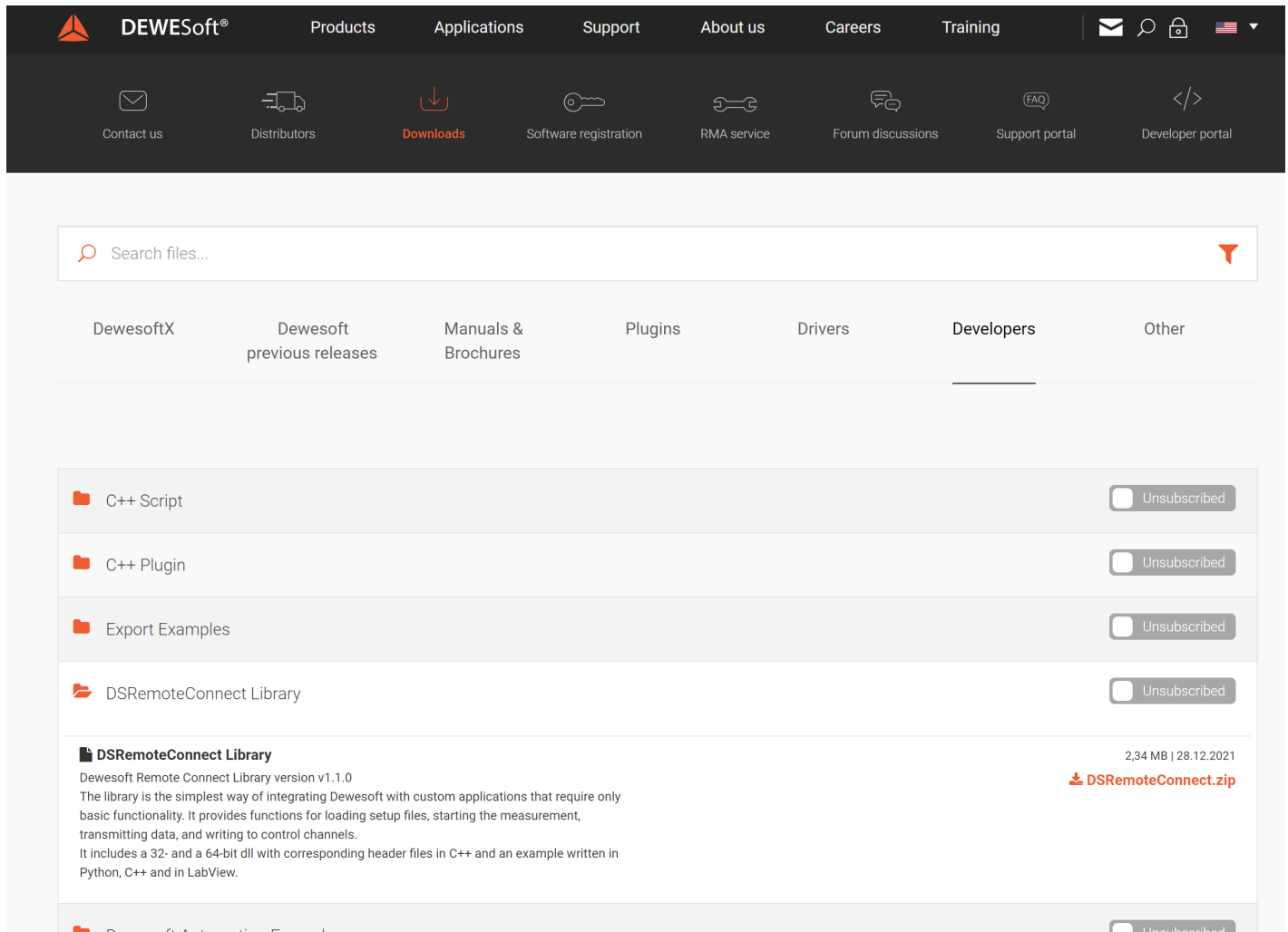
# Second way of enumerating Channels
labelsPtr = pointer((POINTER(ctypes.c_char_p) * numChannels.value)())
localValue = c_size_t(numChannels.value)
hllDll.dsconEnumerateChannels(dsconInstance, labelsPtr, byref(localValue))
name = (c_char * 50)()
ch_name = ctypes.cast(name, c_char_p)
unit = (c_char * 50)()
ch_unit = ctypes.cast(name, c_char_p)
ch_instances = []
for i in range(0, numChannels.value):
    name = (c_char * len(labelsPtr[0][0][i]))()
    ch_index = ctypes.cast(name, c_char_p)
```

# Introduction to DSRemote library

Do you ever want to control DewesoftX but you found out that DCOM (short for Distributed Component Object Model) is hard to understand and Dewesoft NET protocol even harder? With the help of C DLL (short for dynamic link library) named DSRemote you will be able to control and get data from the DewesoftX as easy as never before.

# How to install the DSRemote library

To install DSRemote you need to go to our page. Under the development section, you need to find DSRemote. Note that you have to be logged in to access the developer section.



The screenshot shows the DEWESoft website's navigation menu and a search results page. The navigation menu includes: DEWESoft®, Products, Applications, Support, About us, Careers, Training, Contact us, Distributors, Downloads (highlighted), Software registration, RMA service, Forum discussions, Support portal, and Developer portal. Below the navigation is a search bar with the text "Search files...". Underneath the search bar are tabs for: DewesoftX, Dewesoft previous releases, Manuals & Brochures, Plugins, Drivers, Developers (selected), and Other. The main content area displays a list of items with "Unsubscribed" buttons:

- C++ Script
- C++ Plugin
- Export Examples
- DSRemoteConnect Library

The "DSRemoteConnect Library" item is expanded, showing the following details:

- DSRemoteConnect Library**
- 2,34 MB | 28.12.2021
- [DSRemoteConnect.zip](#)
- Dewesoft Remote Connect Library version v1.1.0
- The library is the simplest way of integrating Dewesoft with custom applications that require only basic functionality. It provides functions for loading setup files, starting the measurement, transmitting data, and writing to control channels.
- It includes a 32- and a 64-bit dll with corresponding header files in C++ and an example written in Python, C++ and in LabView.

After downloading, you will have a zip that contains examples written in python, C++ and in Labview. Version 1.1.0 or newer is needed in case you want to use Dewesoft NET option.

# Overview of DSRemote

DSRemote is a DLL written in the C programming language. This DLL can be used in any programming language as long as it has support for C type DLLs. The data gotten from the DewesoftX can be used in post-analysis software or can be used to store data in databases. DSRemote can use two different protocols to communicate with DewesoftX. They are called DCOM and Dewesoft NET protocol. DCOM is a proprietary Microsoft technology for communication between software components. This technology is used in Dewesoft extensions and Sequencer as the backend for communication with DewesoftX. Dewesoft NET protocol on the other hand consists of two network protocols. To control DewesoftX before the measurement the Telnet protocol is used. After we switch to measurement DewesoftX starts to send us the data in TCP/IP stream of data.

To use DSRemote there are some requirements.

- You need to use the DewesoftX version **2022.1** or higher.
- It's recommended that the last version of C++ redistributable be installed on the computer.
- In case you are using the Dewesoft NET protocol you need a license for it.

For better performance and when having distributed system, it is recommended to use the Dewesoft NET protocol even in case you are reading data from the same computer (local connection).

---

Which functions does the DSRemote cover:

- Loading setup
  - Getting basic properties from the channel (unit or name)
  - Hiding or showing DewesoftX UI
  - Start/Stop measurement
  - Reading data from the channels
  - Writing values to control channels
-

Now we will take a look at how these functions are implemented in the DSRemote DLL. Before we can call functions from the DLL you need to make an instance. To make this instance you need to call function.

```
int dsconCreateInstance(DSXInstanceHandle* handle, int connectionType);
```

As the first parameter, you send a handle by the reference. This handle presents a connection to DewesoftX software. As the second parameter you define what kind of connection you want to make to the DewesoftX software. For now, as stated before we support two types of connection. DCOM and Dewesoft NET. In the DLL they are defined as

```
typedef enum {  
    DSX_CONNECTION_TYPE_DCOM = 0,  
    DSX_CONNECTION_TYPE_NET = 1  
} DSX_CONNECTION_TYPE;
```

Where 0 presents DCOM connection and 1 NET connection. To make a NET connection we call the function like.

```
dsconCreateInstance(dsconInstance, DSX_CONNECTION_TYPE_NET)
```

After calling this function we will get an instance that will be used to control DewesoftX. With acquired instances, we can call other functions. Before we can fully control DewesoftX we first need to connect to it. To connect to DewesoftX we need to call

```
int dsconConnect(DSXInstanceHandle handle, char* connectionString)
```

Where the first parameter will be an instance that we got from the dsconCreateInstance and the second parameter is the string that gives us the information about the connection. In case we will use DCOM protocol this parameter can be empty. For the Dewesoft NET protocol, this parameter consists of information delimited with a colon. This information presents the address that we will connect over the ethernet. For example, if we want to connect to a computer with IP 10.2.10.254 that uses for Telnet communication port 8999 and for data transmission port 8001 then we define a string as

```
œ10.2.10.254:8999:8001œ
```

About these settings, you can read more about them in the pro tutorial for [Dewesoft NET](#) or our [manual](#).

After we are connected to DewesoftX we are ready to do things.

# Example - reading and writing data over NET

Now let's take a look at an example

```
from ctypes import *
import ctypes from ctypes.wintypes import *
import time
import numpy as np

def doErrCheck(err_code, descr):
    if err_code < 0:
        print(str(descr) + " errorCode: " + str(err_code))
    return err_code

hllDll = WinDLL
("C:\\DXEProjects\\Tools\\DSRemoteConnect\\DSRemoteConnect\\Debug\\DSRemoteConnect64.dll")
dsconInstance = HANDLE()
doErrCheck(hllDll.dsconCreateInstance(pointer(dsconInstance), 1), "dsconCreateInstance")
charArray = create_string_buffer("10.2.120.255:8999:8001".encode())
doErrCheck(hllDll.dsconConnect(dsconInstance, charArray), "dsconConnect")
numChannels = c_size_t()
doErrCheck(hllDll.dsconGetChannelCount(dsconInstance, byref(numChannels)),
"dsconGetChannelCount")
print(numChannels.value)

# Second way of enumerating Channels
labelsPtr = pointer((POINTER(ctypes.c_char_p) * numChannels.value)()) localValue =
c_size_t(numChannels.value)
hllDll.dsconEnumerateChannels(dsconInstance, labelsPtr, byref(localValue))
name = (c_char * 50)()
ch_name = ctypes.cast(name, c_char_p)
unit = (c_char * 50)()
ch_unit = ctypes.cast(name, c_char_p)
ch_instances = []

for i in range(0, numChannels.value):
    name = (c_char * len(labelsPtr[0][0][i]))()
    ch_index = ctypes.cast(name, c_char_p)
    ch_index.value = labelsPtr[0][0][i]
    ch_instance = HANDLE()
    hllDll.dsconCreateChannelInstance(dsconInstance, ch_index, pointer(ch_instance))
    ch_instances.append(ch_instance)
    hllDll.dsconChannelGetName(ch_instances[i], ch_name, 50)
    print(ch_name.value)
    hllDll.dsconGetChUnit(ch_instances[i], ch_unit, 50)
    print(ch_unit.value)

doErrCheck(hllDll.dsconChannelSetTransferred(ch_instances[len(ch_instances) - 1], False),
'dsconChannelSetTransferred')
doErrCheck(hllDll.dsconStartMeasurement(dsconInstance), "dsconStartMeasurement")
data2 = ctypes.cast((c_double * 100000)(), POINTER(c_double))
time_stamps2 = ctypes.cast((c_double * 100000)(), POINTER(c_double))
count = 0
```

```

while True:
    try:
        countData2 = c_size_t(100000)
        for i in range(len(ch_instances)):
            doErrCheck(hIIDll.dsconChannelReadScalarData_2(ch_instances[i], data2, time_stamps2,
byref(countData2)), "dsconChannelReadScalarData_2"
            ^ ^ ^ ^
            doErrCheck(hIIDll.dsconControlChannelWriteData(ch_instances[1], c_double(0.69 + count)), "dsconC
            doErrCheck(hIIDll.dsconControlChannelWriteData(ch_instances[2], c_double(0.70 + count)), "dsconContrc
            doErrCheck(hIIDll.dsconControlChannelWriteData(ch_instances[3], c_double(0.71 + count)), "dsconContrc
            count += 1
    except:
        print("While interrupted")
        break

print("Finally")
doErrCheck(hIIDll.dsconStopMeasurement(dsconInstance), "dsconStopMeasurement")
doErrCheck(hIIDll.dsconDisconnect(dsconInstance), "") # this will close Dewesoft.exe and clear the DCOM
for i in range(0, numChannels.value - 1):
    doErrCheck(hIIDll.dsconFreeChannelInstance(ch_instances[i]), "dsconFreeChannelInstance") # this will
free chnl instances of dll
doErrCheck(hIIDll.dsconDestroyInstance(dsconInstance), "dsconDestroyInstance") # this will free dll
instance

```

The example is written in python. This example uses Dewesoft Net as protocol and firstly outputs the channels that are present in the setup and outputs their name and unit. After listing information, we go into measure mode and read the data. After the data is read, we write some values to control the channel. Now let's look at the code by sections.

In the following section, we are listing the channels and outputting the channel name and unit.



```

numChannels = c_size_t()
doErrCheck(hIIDll.dsconGetChannelCount(dsconInstance, byref(numChannels)),
"dsconGetChannelCount")
print(numChannels.value)

# Second way of enumerating Channel
labelsPtr = pointer((POINTER(ctypes.c_char_p) * numChannels.value)())
localValue = c_size_t(numChannels.value)
hIIDll.dsconEnumerateChannels(dsconInstance, labelsPtr, byref(localValue))
name = (c_char * 50)()
ch_name = ctypes.cast(name, c_char_p)
unit = (c_char * 50)()
ch_unit = ctypes.cast(name, c_char_p)
ch_instances = []

for i in range(0, numChannels.value):
    name = (c_char * len(labelsPtr[0][0][i]))()
    ch_index = ctypes.cast(name, c_char_p)
    ch_index.value = labelsPtr[0][0][i]
    ch_instance = HANDLE()
    hIIDll.dsconCreateChannelInstance(dsconInstance, ch_index, pointer(ch_instance))
    ch_instances.append(ch_instance)
    hIIDll.dsconChannelGetName(ch_instances[i], ch_name, 50)
    print(ch_name.value)
    hIIDll.dsconGetChUnit(ch_instances[i], ch_unit, 50)
    print(ch_unit.value)
    print(ch_unit.value)

```

Before we can loop over all the channels we need to find out how many channels are there. To get this information we need to call the function

```
int dsconGetChannelCount(DSXInstanceHandle handle, size_t* count);
```

As the first parameter, we pass the instance that we described in the [section](#). As the second parameter, we by reference pass the value. After calling this function, the number of channels that are present in the DewesoftX will be stored in this variable. Before we can access the properties we need to get the unique ID that presents a channel. To get the unique ID's we call the function.

```
int dsconEnumerateChannels(DSXInstanceHandle handle, ChannelIdList* channelIdList, size_t* count);
```

This will return values by reference. The second parameter will return an array of strings, each element representing unique ID for each DewesoftX channel. The last parameter returns how many channels are there present in the array. Unique IDs are used to create DewesoftX channel instances. To create a channel instance we call function.

```
int dsconCreateChannelInstance(DSXInstanceHandle handle, ChannelID id, ChannelInstanceHandle* chInstance);
```

The channel instance is returned as the third parameter by reference. With the acquired channel instance we can get info

about channel name and channel unit. To get the channel name we call the function

```
int dsconChannelGetName(ChannellInstanceHandle channel, char* name, size_t len);
```

and to get the channel unit we call the function

```
int dsconGetChUnit(ChannellInstanceHandle handle, char* resultUnit, size_t len);
```

After each call, we print it to standard output.

In the next section, we will take a look at how to read and write the data to channels. Let's take a look at the section.

```
doErrCheck(hIIDll.dsconStartMeasurement(dsconInstance), "dsconStartMeasurement")
data2 = ctypes.cast((c_double * 100000)(), POINTER(c_double))
time_stamps2 = ctypes.cast((c_double * 100000)(), POINTER(c_double))
count = 0

while True:
    try:
        countData2 = c_size_t(100000)
        for i in range(len(ch_instances)):
            doErrCheck(hIIDll.dsconChannelReadScalarData_2(ch_instances[i], data2, time_stamps2,
byref(countData2)), "dsconChannelReadScalarData_2")

            doErrCheck(hIIDll.dsconControlChannelWriteData(ch_instances[1], c_double(0.69 + count)),
"dsconControlChannelWriteValue1")
            doErrCheck(hIIDll.dsconControlChannelWriteData(ch_instances[2], c_double(0.70 + count)),
"dsconControlChannelWriteValue2")
            doErrCheck(hIIDll.dsconControlChannelWriteData(ch_instances[3], c_double(0.71 + count)),
"dsconControlChannelWriteValue3")
            count += 1

    except:
        print("While interrupted")
        break

print("Finally")
doErrCheck(hIIDll.dsconStopMeasurement(dsconInstance), "dsconStopMeasurement")
doErrCheck(hIIDll.dsconDisconnect(dsconInstance), "") # this will close Dewesoft.exe and clear the DC
for i in range(0, numChannels.value - 1)
    doErrCheck(hIIDll.dsconFreeChannellInstance(ch_instances[i]), "dsconFreeChannellInstance") # this will
free chnl instances of dewesoft
doErrCheck(hIIDll.dsconDestroyInstance(dsconInstance), "dsconDestroyInstance") # this will free dll
instance
```

Before we can start reading or writing the data we first need to go to the measure. To switch to measure mode we need to call the function

```
int dsconStartMeasurement(DSXInstanceHandle handle);
```

After this call, we can read and write the data to channels. This library supports reading the data from any channel but only supports writing to control channels. To know if the channel is the control you need to call the function.

```
int dsconIsChannelControl(ChannelInstanceHandle handle, bool* result);
```

Before we can read the data we need to allocate a local buffer. The samples from the channels will be added to this buffer. If you take a look at python we see in the following code

```
data2 = ctypes.cast((c_double * 100000)(), POINTER(c_double))  
time_stamps2 = ctypes.cast((c_double * 100000)(), POINTER(c_double))
```

That we are preallocating local buffer for data and timestamps. The size is 100000 double values. The size of the local buffer depends on how many values from DewesoftX values will be copied. In case the preallocated buffer is too small, samples in DewesoftX will be lost, because they will be overwritten. The following function:

```
int dsconChannelReadScalarData_2(ChannelInstanceHandle handle, double* data, double* timestamps,  
size_t * count);
```

accepts several parameters. The first one accepts a channel instance which represents the channel from which we will read the data. The second and third parameters accept preallocated buffer as the fourth parameter the function accepts the size of the buffer. In case the number of samples available is lower than the size of the buffer it will change accordingly to the number of added samples.

As pointed before we have also added a function to write to control channels. Function for writing values to control channels looks like this:

```
int dsconControlChannelWriteData(ChannelInstanceHandle handle, double data);
```

Same as in `dsconChannelReadScalarData` the first parameter represents the channel instance that is connected to the DewesoftX channel. The second parameter presents the value that we want to write to the control channel.

After we are done with the measurement we call the function:

```
int dsconStopMeasurement(DSXInstanceHandle handle);
```

After we are done with measurement and want to stop the program, we need to disconnect from the DewesoftX. To do this we call the function:

```
int dsconDisconnect(DSXInstanceHandle handle);
```

After we are disconnected, we also want the software to be closed correctly. The software will be closed correctly if we free all the instances we made during our software. To do this we have these functions:

```
int dsconFreeEnumerateChannels(DSXInstanceHandle handle, ChannelIdList* channelIdList)
int dsconFreeChannelInstance(ChannelInstanceHandle chInstance)
int dsconDestroyInstance(DSXInstanceHandle handle);
```

With `dsconFreeEnumerateChannels` we clean the array of strings that we made during the call of function `dsconEnumerateChannels`. With the function `dsconFreeChannelInstance`, you clean the channel instances that were made with the function `dsconCreateChannelInstance`. After all other instances are freed, we call the last function `dsconDestroyInstance`.