

GETTING STARTED WITH THE NDI™ SDK

©2017 NewTek Inc. 12/8/2017

SENDING VIDEO

We are going to write a very simple video sender that sends a single frame of video. We create an NDI sender instance, although this can be configured manually we can also just allow the SDK to choose all settings manually, so our next list would be:

```
NDIlib_send_instance_t my_ndi_send = NDIlib_send_create();  
If (!my_ndi_send) /* Failed to create sender */return false;
```

We are now going to send an HD frame of BGRA data that we already have rendered in memory at a pointer p_bgra_frame. We now initialize a frame descriptor of the frame that is going to be send with:

```
NDIlib_video_frame_v2_t video_frame_data;  
video_frame_data.xres = xres;  
video_frame_data.yres = yres;  
video_frame_data.FourCC = NDIlib_FourCC_type_BGRA;  
video_frame_data.p_data = p_bgra;
```

Please note that if you look at the definition of this structure you can specify the aspect ratio, fielding mode, line strides, meta-data and much more, but when those capabilities are not required they may simply be ignored.

You may now send this frame to NDI with a single call:

```
NDIlib_send_send_video(my_ndi_send, &video_frame_data);
```

You can send as many frames as you want in this way, on a frame by frame basis you can change any properties including resolution, color format, aspect ratio, etc... You do not need to continually send video frames because NDI will buffer the last one itself in the NDI compressed format so that it can be resent should anyone wish to connect to your NDI sender even when you are not sending any frames. One you wish to exit your application and no longer need the connection you may simply call the destroy function.

```
NDIlib_send_destroy(my_ndi_send);
```

This is all that is needed to implement a simply sender and although there are many other capabilities and features you can take advantage of, this would be an entirely correct application that works fully.

FINDING SOURCES

In the previous example we described how to send video over NDI. In order to discover the list of sources that are currently being sent onto the network you start by creating an NDI finder instance as follows:

```
NDIlib_find_instance_t my_ndi_find = NDIlib_find_create_v2();  
If (!my_ndi_find) /* Failed to create finder */return false;
```

Once you have a finder instance it will immediately start looking for network sources; it might take a few seconds for it to find all of them but at any time you can query it in order to get all the senders that it has found on the network:

```
uint32_t no_srcs; // This will contain how many senders have been found so far.  
const NDIlib_source_t* p_senders = NDIlib_find_get_current_sources(my_ndi_find, &no_srcs);
```

`p_senders` is an array of length `no_srcs` that has all the sources found. The lifetime of the `p_senders` pointer is valid until the next call to `NDIlib_find_get_current_sources` or the finder is destroyed. Because sources on a network do not all get detected at once and new sources might come online and existing sources might go offline you can either repeat the call above each time you need the list, or you can make the following call that allows you to sleep for some period of time until the list of sources has changed.

```
bool source_list_has_changed = NDIlib_find_wait_for_sources(my_ndi_find, 2500/* 2.5 seconds */);
```

Once you no longer want to keep up to date with the available sources on the network you can simply destroy the finder with a call too.

```
NDIlib_find_destroy(my_ndi_find);
```

This is all that is needed to locate sources and know when the list of sources found has changed. Once you have the list of sources that are available to you then you can use one of these to receive video as described in the next section.

VIDEO